

COVER Book Cover

Family Planning and Application Development

--
TeamConnection Unleashed

July 1996

Document Number SG26-2008-00

NOTICES Notices

+--- **Take Note!** -----+

|
| Before using this information and the product it supports, be sure |
| to read the general information in Appendix D, "Special Notices" |
| in topic D.0. |
|
+-----+

EDITION Edition Notice

First Edition (July 1996)

This edition applies to Version 1.0 of TeamConnection, Product Number 31H3744, for use with the OS/2 Warp Operating System.

Comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 471 Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

| Copyright International Business Machines Corporation 1996. All rights reserved.

Note to U.S. Government Users -- Documentation related to restricted rights -- Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

CONTENTS Table of Contents

COVER	Book Cover
NOTICES	Notices
EDITION	Edition Notice
CONTENTS	Table of Contents
FIGURES	Figures
TABLES	Tables
PREFACE	Preface
FRONT_1	The Team That Wrote This Redbook
FRONT_1.1	How This Redbook Is Organized
FRONT_1.2	Comments Welcome
1.0	Chapter 1. Application Development Challenges and TeamConnection
1.1	How Can TeamConnection Help?
1.2	What Does TeamConnection Do?
1.3	TeamConnection Customer Value
2.0	Chapter 2. Terminology
2.1	Configuration Management
2.2	Release Management
2.3	Version Control
2.4	Integrated Build
2.4.1	Build Mechanisms
2.4.2	Distributed Build
2.5	Packaging and Distribution Support
2.5.1	TeamConnection's Packaging Utilities
2.5.2	Your Own Distribution Tool
2.6	Integrated Problem Tracking and Change Control
2.7	Report Facility
2.8	Backup and Recovery
2.9	Repository and Model Support
2.10	Information Model
2.11	Data Constraints
2.12	Information Model Schema Evolution
2.13	Application Development Tools Integration
3.0	Chapter 3. Planning for TeamConnection
3.1	Installation Planning
3.1.1	Hardware Requirements
3.1.2	Software Requirements
3.2	Family Planning
3.3	User IDs and Host Lists
3.4	The Component Hierarchy
3.4.1	Reflecting the Product Organization
3.4.2	Parallel Components
3.4.3	Multiple Parents
3.4.4	Component Evolution
3.4.5	Naming the Components
3.5	Component Ownership, Access Lists, and Notification Lists
3.5.1	Access Lists
3.5.2	Notification Lists
3.6	Releases
3.6.1	Selecting Serial or Concurrent Development
3.6.2	Naming Your Releases
3.6.3	Using Remote Databases
3.7	Planning Your Processes
3.7.1	Planning the Component Process
3.7.2	Planning Your Release Processes
3.8	Planning Your Build Environment
3.9	Choosing a Naming Convention
4.0	Chapter 4. Installing TeamConnection
4.1	Preparing to Install
4.2	Installing the TeamConnection Components
4.2.1	STEP 1: Starting the Installation
4.2.2	STEP 2: Selecting Installation Options
4.2.3	STEP 3: Selecting the Components for Installation
4.2.4	STEP 4: Completing the Installation
4.3	Setting the Environment Variables
4.4	Configuring and Starting the ObjectStore Server
4.5	Verifying the Installation of TeamConnection
4.5.1	STEP 1: Create Your Test Family
4.5.2	STEP 2: Start the Family Server
4.5.3	STEP 3: Verify TeamConnection Installation
4.6	Starting the TeamConnection Client
5.0	Chapter 5. Creating the Family
5.1	Family Information Page
5.1.1	Fields
5.2	Authority Groups Page
5.2.1	Fields
5.2.2	Push Buttons
5.3	The Interest Groups Page
5.3.1	Fields
5.3.2	Push Buttons
5.4	Configurable Fields Page
5.4.1	Fields
5.4.2	Push Buttons
5.5	Component Processes Page

5.5.1	Fields
5.5.2	Push Buttons
5.6	Release Processes Page
5.6.1	Fields
5.6.2	Push Buttons
5.7	User Exits Page
5.7.1	Fields
5.7.2	Push Buttons
5.8	Family Servers Window
6.0	Chapter 6. Using TeamConnection
6.1	Starting the TeamConnection Client
6.1.1	TeamConnection - Tasks Window
6.1.2	The Settings Notebook
6.1.2.1	Environment Page
6.1.2.2	Setup Page
6.1.2.3	GUI Page
6.1.2.4	Extract Page
6.1.2.5	Pool Page
6.2	Creating TeamConnection Users and Host Lists
6.2.1	Using the GUI
6.2.2	Using the Command Line Interface
6.3	Creating Components
6.3.1	Using the GUI
6.3.2	Transfer Component Ownership
6.3.3	Using the Command Line Interface
6.4	Creating a Release
6.4.1	Using the GUI
6.4.2	Using the Command Line Interface
6.5	Creating a Work Area
6.5.1	Using the GUI
6.5.2	Using the Command Line Interface
6.6	Creating Parts
6.6.1	Common Part
6.6.2	Using the GUI
6.6.3	Using the Command Line Interface
6.7	Working with Parts
6.7.1	Check Out a Part
6.7.2	Edit a Part
6.7.3	Check in a Part
6.7.4	Lock a Part
6.7.5	Unlock a Part
6.7.6	Extract a Part
6.8	Creating Versions
7.0	Chapter 7. Using TeamConnection to Build Applications
7.1	Build Users and Their Roles
7.2	Working with Build Scripts
7.2.1	For OS/2
7.2.2	For MVS
7.2.2.1	Extended JCL Syntax
7.2.2.2	Passing Parameters
7.2.2.3	Sample Build Scripts
7.2.3	For Parsers
7.3	Organizing Build Pools and Build Environments
7.4	Organizing the Build Agents and Build Processors
7.4.1	Starting Your Build Server
7.4.1.1	Starting a Build Processor on OS/2
7.4.1.2	Starting a Build Processor on MVS
7.4.1.3	Starting a Build Agent
7.5	Creating a Builder
7.6	Creating a Parser
7.7	Creating a Build Tree
7.8	Using a NULL Builder
7.9	Building an Application
8.0	Chapter 8. Using TeamConnection to Package Products
8.1	Setting Up Your Build Tree for Packaging
8.1.1	Set Up a Build Tree for the Gather/2 Tool
8.1.2	Set Up a Build Tree for the NVBridge/2 Tool
8.1.3	Set Up a Build Tree for Other Distribution Tools
8.2	Using the Gather/2 Tool
8.2.1	Using the teamcpak Command
8.2.1.1	Command Line Flags
8.2.1.2	Examples of the teamcpak Gather Command
8.2.2	Writing a Package File
8.2.2.1	Syntax Rules
8.3	Using the NVBridge/2 Tool
8.3.1	Using the teamcpak Command
8.3.1.1	Command Line Flags
8.3.1.2	Examples of the teamcpak nvbridge Command
8.3.2	Writing a Package File
8.3.2.1	Syntax Rules
8.3.2.2	Keywords for an NVBridge/2 Package File
8.3.3	Problem Determination
8.3.4	NVBridge Utilities
9.0	Chapter 9. Using TeamConnection's Integrated Problem Tracking and Change Control

9.1	The Roles People Play
9.1.1	Commands They Can Use
9.1.2	Actions They Can Perform
9.2	Working with Defects and Features
9.2.1	Defect and Feature States
9.2.1.1	Design, Size, Review
9.2.2	A Sample Feature
9.2.2.1	Opening a Feature
9.2.2.2	The Design State
9.2.2.3	The Size State
9.2.2.4	The Review State
9.2.2.5	Accepting the Feature
9.2.2.6	The Working State
9.2.2.7	The Verify State
9.2.2.8	The Closed State
9.3	Change Control
9.3.1	Work Area States
9.3.2	Driver States
9.3.3	Using the Preship Process
9.3.3.1	Opening a Defect
9.3.3.2	Moving a Defect to the Design State
9.3.3.3	Moving a Defect to the Size State
9.3.3.4	Moving a Defect to the Review State
9.3.3.5	Accepting the Defect
9.3.3.6	The Defect in the Working State
9.3.3.7	The Work Area in the Approval State
9.3.3.8	The Work Area in the Fix State
9.3.3.9	Integrating the Work Area
9.3.3.10	Creating a Driver and Adding a Driver Member
9.3.3.11	Committing the Driver
9.3.3.12	Completing the Driver
9.3.3.13	Completing Test Records
9.3.3.14	Completing the Verification Records
9.4	Summary
10.0	Chapter 10. Advanced Topics
10.1	Concurrent Development
10.1.1	Using Concurrent Development with No Tracking Process
10.1.2	Using Concurrent Development with a Tracking Process
10.1.3	Reconciling Collision Records
10.1.4	Using the Merge Tool
10.2	Using Configurable Fields
10.2.1	Defining Configurable Field Types
10.2.2	Changing or Creating Configurable Fields
10.2.3	Displaying Configurable Field Properties
10.3	Using Report Facilities
10.4	User Exits
10.4.1	Writing User Exit Programs
10.4.2	Suggestions
11.0	Chapter 11. Maintaining Your TeamConnection Environment
11.1	Changing the Age of Defects and Features
11.1.1	The Age Utility
11.1.2	The resetAge Utility
11.2	Resolving TeamConnection Errors
11.2.1	Using the Error Log
11.2.2	Using the Audit Log
11.2.2.1	Cleaning up the Audit Log
11.2.3	Using the Trace Facility
11.3	Maintaining the TeamConnection Database
11.3.1	osbackup
11.3.2	oscp
11.3.3	osrestor
11.3.4	ossvrpin
11.3.5	ossvrsta
11.3.6	osverify
12.0	Chapter 12. Family Administration
12.1	Creating or Modifying Authority Groups
12.1.1	Editing the authorit.ld File
12.1.2	Reloading the Authority Table
12.1.3	Using the GUI
12.2	Creating or Modifying Interest Groups
12.2.1	Editing the interest.ld File
12.2.2	Reloading the Interest Table
12.2.3	Using the GUI
12.3	Configuring Component or Release Processes
12.3.1	Editing the comproc.ld and relproc.ld Files
12.3.2	Reloading the Configurable Process Tables
12.3.3	Using the GUI for Component Processes
12.3.4	Using the GUI for Release Processes
12.4	Defining Configurable Field Types
12.4.1	Manually Defining Configurable Field Types
12.4.1.1	Reloading the Config Table
12.4.2	Creating and Modifying Configurable Fields Using the chfield Command
12.4.2.1	Creating Configurable Fields
12.4.2.2	Creating Configurable Fields by Copying from Another Family

12.4.2.3	Updating Configurable Fields
12.4.3	Working with Configurable Fields Using the GUI
12.4.3.1	Creating and Modifying Configurable Fields
12.4.3.2	Working with the Defect Pages
12.4.3.3	Working with the Feature Pages
12.4.3.4	Working with the Part Pages
12.4.3.5	Working with the User Pages
12.5	Changing Report Formats
12.5.1	Manually Changing Report Formats
12.5.2	Changing Report Formats by Using the GUI
12.6	Setting Up User Exits
12.6.1	Editing the userExit File
12.6.2	Using the GUI
13.0	Chapter 13. Software Configuration Management and Change Management
13.1	Why You Need Them
13.2	The Goals
13.3	The Formal Definition
13.4	What They Do
13.5	Who Needs Them
13.5.1	Big Development Efforts
13.5.2	Medium-Sized Development Efforts
13.5.3	Small Development Efforts
13.6	Interaction with Development Methodologies
13.7	Interaction with Project Management
13.8	Interaction with Quality Assurance
13.9	Configuration Management History and Statistics
14.0	Chapter 14. Implementation of ISO 9001 Using TeamConnection
14.1	ISO 9000
14.2	TeamConnection and ISO 9001
14.2.1	Document Control
14.2.2	Version Control in ISO 9001
14.2.2.1	Design Control
14.2.2.2	Product Identification and Traceability
14.2.2.3	Inspection and Test Status
14.2.2.4	Control of Nonconforming Product
14.2.3	Internal Quality Audits
14.3	Conclusion
14.4	Brief Description of ISO 9000-3
14.4.1	Configuration Management
14.4.1.1	Configuration Identification and Traceability
14.4.1.2	Change Control
14.4.1.3	Configuration Status Report
14.4.2	Design Control
14.4.3	Document Control
14.5	References
A.0	Appendix A. Component Hierarchy Creation Aid
B.0	Appendix B. Build Tree Aid
C.0	Appendix C. Useful REXX Programs
C.1	A Program to Re-create the Family
C.2	A Program to Start the Family
C.3	A Program to Check In Multiple Parts
C.4	A Program to Check Out Multiple Parts
C.5	A Program to Create Multiple Parts
C.6	A Program to Extract Multiple Parts
D.0	Appendix D. Special Notices
E.0	Appendix E. Related Publications
E.1	International Technical Support Organization Publications
E.2	Other Publications
E.2.1	TeamConnection
E.2.2	NetView DM/2
E.2.3	ObjectStore Release 4.0 Publications
E.2.4	IBM OS/2 Publications
BACK_1	How to Get ITSO Redbooks
BACK_1.1	How IBM Employees Can Get ITSO Redbooks
BACK_1.2	How Customers Can Get ITSO Redbooks
BACK_1.3	IBM Redbook Order Form
GLOSSARY	Glossary
ABBREVIATION	List of Abbreviations
INDEX	Index

FIGURES Figures

1.	TeamConnection Relationship among Objects	2.0
2.	The Family Component Hierarchy	2.1
3.	Access List	2.1
4.	Notification List	2.1
5.	Managed Objects	2.1
6.	Release Management	2.2
7.	Versioning	2.3
8.	Work Area	2.3
9.	Integrated Build	2.4
10.	Build Tree, Build Event, Builder, Build Script, and Parser	2.4.1
11.	Distributed Build	2.4.2
12.	Build Process Packaging Steps	2.5
13.	Electronic Distribution	2.5.1
14.	Problem Tracking	2.6
15.	Driver	2.6
16.	Information Model Views	2.9
17.	Information Model	2.10
18.	Application Development Tools Integration	2.13
19.	Component Hierarchy Models	3.4
20.	Releases	3.6
21.	Processes	3.7
22.	States	3.7
23.	Component Processes	3.7.1
24.	Release Processes	3.7.2
25.	The TeamConnection Family Administrator Window	5.0
26.	Family Information Page	5.1.1
27.	Authority Groups Page	5.2.2
28.	Interest Groups Page	5.3.2
29.	Configurable Fields Page	5.4.2
30.	Component Processes Page	5.5.2
31.	Release Processes Page	5.6.2
32.	User Exits Page	5.7.2
33.	Family Servers Window	5.8
34.	TeamConnection - Tasks Window	6.1.1
35.	Settings Window	6.1.2
36.	Information Window for Testing Connection to Server	6.1.2.1
37.	Setup Page	6.1.2.2
38.	GUI Page	6.1.2.3
39.	Extract Page	6.1.2.4
40.	Pool Page	6.1.2.5
41.	User Filter Window	6.2.1
42.	TeamConnection - Users Window	6.2.1
43.	Create User Window	6.2.1
44.	TeamConnection - Users Window with Pop-Up Menu	6.2.1
45.	Add Host Window	6.2.1
46.	TeamConnection - Host Lists Window	6.2.1
47.	Creating Users through the Command Line Interface	6.2.2
48.	Creating a Host List through the Command Line Interface	6.2.2
49.	Component Filter Window	6.3.1
50.	Create Components Window	6.3.1
51.	Our Component Structure	6.3.1
52.	The Components Window	6.3.1
53.	Modify Component Owner Window	6.3.2
54.	Creating Components through the Command Line Interface	6.3.3
55.	Creating Components with Different Owners	6.3.3
56.	Releases Filter Window	6.4.1
57.	Create Releases Window	6.4.1
58.	TeamConnection - Releases Window	6.4.1
59.	Creating Releases through the Command Line Interface	6.4.2
60.	Create Work Areas Window	6.5.1
61.	Work Area Filter Window	6.5.1
62.	TeamConnection - Work Areas Window	6.5.1
63.	Creating Work Areas through the Command Line Interface	6.5.2
64.	Create Parts Window	6.6.2
65.	Creating Parts through the Command Line Interface	6.6.3
66.	TeamConnection - Parts Window	6.7
67.	The TeamConnection - PartFull Window	6.7
68.	Check Out Parts Window	6.7.1
69.	Parts Window for Same Work Area but Different User	6.7.1
70.	Parts Window for Different Work Area and User	6.7.1
71.	TeamConnection - PartFull Window	6.7.1
72.	Medit REXX Command File	6.7.2
73.	Edit Part Window	6.7.2
74.	Finished Editing Part Window	6.7.2
75.	Pop Up Choices for Checked-Out Part	6.7.3
76.	Check In Parts Window	6.7.3
77.	Check In Parts Error Window	6.7.3
78.	Selecting Break Common Link in the Check In Parts Window	6.7.3
79.	Lock Parts Window	6.7.4
80.	Edit Part Information Window	6.7.4
81.	Unlock Parts Window	6.7.5
82.	Extract Parts Window	6.7.6
83.	Freeze Work Areas Window	6.8

84.	Integrate Work Areas Window	6.8
85.	Sample Build Script for C++ Compile on OS/2	7.2.1
86.	MVS JCL for C Compile	7.2.2
87.	Build Topology	7.4
88.	The RUNPGM JCL	7.4.1.2
89.	The RUNPGMT JCL	7.4.1.2
90.	Create Builder Window	7.5
91.	Create Parser Window	7.6
92.	Build Tree	7.7
93.	TeamConnection - BuildView Window	7.7
94.	Application Build Using a NULL Builder	7.8
95.	Application Build	7.9
96.	Build Parts Window	7.9
97.	Build Progress Window	7.9
98.	Build Progress Window after Completed Build	7.9
99.	TeamConnection - BuildView Window after Completed Build	7.9
100.	Electronic Distribution Using Gather/2 and NVBridge/2	8.0
101.	Part of the Build Tree for SerataApplication	8.1.1
102.	Adding the Gather/2 Step to the Build Tree	8.1.1
103.	Adding the NVBridge/2 Step to the Build Tree	8.1.2
104.	Using Gather/2	8.2
105.	The NVBridge/2 Software Distribution Model	8.3
106.	Problem Tracking and Integrated Problem Tracking and Change Control	9.0
107.	Defect and Feature Relationships	9.2
108.	Defect and Feature States	9.2.1
109.	Action-State Diagram	9.2.1
110.	Open Feature Window	9.2.2.1
111.	Open Feature Information Window	9.2.2.1
112.	The Features Window	9.2.2.1
113.	Design Features Window	9.2.2.2
114.	Features Window	9.2.2.2
115.	Size Features Window	9.2.2.3
116.	Create Sizing Records Window	9.2.2.3
117.	TeamConnection - Sizing Records Window	9.2.2.3
118.	Accept Sizing Records Window	9.2.2.3
119.	Review Features Window	9.2.2.4
120.	Accept Features Window	9.2.2.5
121.	Verify Features Window	9.2.2.7
122.	Accept Verification Records Window	9.2.2.7
123.	TeamConnection - Features Window after Feature Has Been Closed	9.2.2.8
124.	Defect and Feature Work Area States	9.3.1
125.	Action-State Diagram for Work Areas	9.3.1
126.	Adding Driver Members	9.3.2
127.	Work Area and Driver States	9.3.2
128.	Defect and Feature, Work Area, and Driver State Transitions	9.3.3
129.	Action-State Diagram for the Driver Subprocess	9.3.3
130.	Open Defect Window	9.3.3.1
131.	Open Defect Information Window	9.3.3.1
132.	TeamConnection - Defects Window	9.3.3.1
133.	Design Defects Window	9.3.3.2
134.	Size Defects Window	9.3.3.3
135.	Create Sizing Records Window	9.3.3.3
136.	Sizing Records Window	9.3.3.3
137.	Accept Sizing Records Window	9.3.3.3
138.	Review Defects Window	9.3.3.4
139.	Accept Defects Window	9.3.3.5
140.	TeamConnection - Work Areas Window	9.3.3.5
141.	Approval Records Window	9.3.3.7
142.	Accept Approval Records Window	9.3.3.7
143.	Approval Records Window after Accept	9.3.3.7
144.	TeamConnection - Work Areas Window	9.3.3.8
145.	Complete Fix Records Window	9.3.3.9
146.	Integrate Work Areas Window	9.3.3.9
147.	Create Drivers Window	9.3.3.10
148.	TeamConnection - Drivers Window	9.3.3.10
149.	Add Driver Members Window	9.3.3.10
150.	TeamConnection - Drivers Window after Driver Has Moved to Integrate State	9.3.3.10
151.	Commit Drivers Window	9.3.3.11
152.	TeamConnection - Work Areas Window after Driver Has Been Committed	9.3.3.11
153.	Complete Drivers Window	9.3.3.12
154.	TeamConnection - Work Areas Window after Driver Has Been Completed	9.3.3.12
155.	TeamConnection - Test Records Window	9.3.3.13
156.	Accept Test Records Window	9.3.3.13
157.	TeamConnection - Work Areas Window after All Test Records Have Been Completed	9.3.3.13
158.	TeamConnection - Defects Window after the First Work Area Has Moved to the Complete State	9.3.3.13
159.	Accept Verification Records Window	9.3.3.14
160.	TeamConnection - Defects, Work Areas, and Drivers Windows after	

Defect Has Been Closed	9.3.3.14
161. Concurrent Development	10.1
162. Concurrent Development with No Track Process	10.1.1
163. Concurrent Development with a Track Process	10.1.2
164. Work Area Serata-PWA1 and Work Area Serata-PWA2	10.1.3
165. Integrate Work Areas Information Window	10.1.3
166. Refresh Work Areas Window	10.1.3
167. Refresh Work Areas Information Window	10.1.3
168. TeamConnection - Collision Records Window	10.1.3
169. Reconcile Collision Record Window	10.1.3
170. Merge Tool Main Window	10.1.4
171. Using the Merge Tool	10.1.4
172. Finished Merging Part Window	10.1.4
173. TeamConnection - Collision Records Window after Reconciliation	10.1.4
174. TeamConnection - Work Areas Window after Integration of Serata-PWA2	10.1.4
175. Sample audit.log File	11.2.2
176. The chfield Display with Configurable Field Information	12.4.2.1
177. Configurable Fields Defect Page	12.4.3.2
178. Configurable Fields Feature Page	12.4.3.3
179. Configurable Fields Part Page	12.4.3.4
180. Configurable Fields User Page	12.4.3.5
181. Sample Report Format after Adding Configurable Fields	12.5.1
182. Stanza View Format Page	12.5.2
183. Table View Format Page	12.5.2
184. Why Configuration Management and Change Management Are Necessary	13.0
185. Development Process Relationships	13.9
186. The rct_fam.cmd Program	C.1
187. The strt2008.cmd Program	C.2
188. The bldagnts.lst File	C.2
189. The bldprcs.lst File	C.2
190. The tcchkin.cmd Program	C.3
191. The tcchkout.cmd Program	C.4
192. The tcparts.cmd Program	C.5
193. The tcxtract.cmd Program	C.6

TABLES Tables

1. TeamConnection Server Hardware Requirements 3.1.1
2. Build Processor Hardware Requirements 3.1.1
3. Build Agent Hardware Requirements 3.1.1
4. TeamConnection Client Hardware Requirements 3.1.1
5. TeamConnection Clients and Servers Software Requirements 3.1.2
6. TeamConnection Field Name Lengths 3.9
7. Component Hierarchy Creation Aid 6.3.1
8. TeamConnection Trace Environment Variables 11.2.3
9. Component Hierarchy Creation Aid A.0
10. Build Tree Aid B.0

PREFACE Preface

This book is unique in its detailed coverage of TeamConnection. It focuses on the implementation and use of IBM's TeamConnection and positions the product within the application framework. It also considers the concept of configuration management and version control and explains how they are implemented in TeamConnection.

This book is written for application development managers, project leaders, product evaluators, service providers, family administrators, system administrators, developers, and team leaders. Some knowledge of application development and configuration management is assumed.

The phrase *Family Planning* in the title refers to the TeamConnection database, which is called a *family*. One of the first steps in working with TeamConnection is to plan the *family* database, hence the phrase *Family Planning*.

In writing this book we tried to cover as much as possible of what TeamConnection has to offer. Even though the book contains some 539 pages, we would also like to refer you to the ITSO Redbook, *TeamConnection and WorkFrame Integration Survival Guide*, SG24-4610.

This book comes with "ready-to-run" REXX programs that are intended to be used as helpful tools when working with TeamConnection. The REXX programs and a *time-limited* version of TeamConnection are also included on a CD-ROM that accompanies this book. We hope that you find this book valuable in your efforts to *unleash* the power of TeamConnection.

Enjoy reading.

A handwritten signature in black ink, appearing to read 'Leif Trulsson'. The signature is stylized with a large, sweeping 'S' shape at the beginning and a long, horizontal stroke at the end.

Leif Trulsson
ITSO - San Jose, California
July 1996

FRONT_1 The Team That Wrote This Redbook

This book was written by **Leif G. Trulsson**, a Senior Information Technology Consultant at IBM's International Technical Support Organization-San Jose Center. Leif is an application development specialist with more than 18 years of experience in developing applications for both customers and IBM. He writes extensively and teaches IBM classes worldwide on IBM's team development environment and TeamConnection in particular. Before joining the ITSO in 1994, Leif worked as a Consultant in the Software and Services department, IBM Sweden, where he specialized in client/server applications and application migration issues.

Leif can be reached at:

□ ltrulsson@vnet.ibm.com

Thanks to the following people for their invaluable contributions to this project:

Maggie Cutler
Stephanie Manning and Elizabeth Rice
Elsa Barron
Mary Comianos
Alan Tippet
International Technical Support Organization, San Jose Center

Subtopics

FRONT_1.1 How This Redbook Is Organized

FRONT_1.2 Comments Welcome

FRONT_1.1 How This Redbook Is Organized

This redbook contains 539 pages. It is organized as follows:

- Chapter 1, "Application Development Challenges and TeamConnection"

This chapter provides an introduction to TeamConnection and explains how TeamConnection can support your development efforts.
- Chapter 2, "Terminology"

This chapter provides an overview of TeamConnection terminology and concepts.
- Chapter 3, "Planning for TeamConnection"

This chapter provides useful information about planning for TeamConnection.
- Chapter 4, "Installing TeamConnection"

This chapter provides information about installing TeamConnection.
- Chapter 5, "Creating the Family"

This chapter provides information about creating a TeamConnection database (also known as a family).
- Chapter 6, "Using TeamConnection"

This chapter provides information about using TeamConnection as a first-time user.
- Chapter 7, "Using TeamConnection to Build Applications"

This chapter provides information about building application with the help of TeamConnection and its integrated build function.
- Chapter 8, "Using TeamConnection to Package Products"

This chapter provides information about packaging and distributing applications that are ready for production.
- Chapter 9, "Using TeamConnection's Integrated Problem Tracking and Change Control"

This chapter explains how to take full advantage of TeamConnection's ability to track problems and enhancements and control the changes of parts.
- Chapter 10, "Advanced Topics"

This chapter provides information about such issues as concurrent (parallel) development, configurable fields, and user exits.
- Chapter 11, "Maintaining Your TeamConnection Environment"

This chapter provides information about maintaining the TeamConnection environment.
- Chapter 12, "Family Administration"

This chapter provides useful information for the family administrator and the tasks involved in what is called *family administration*.
- Chapter 13, "Software Configuration Management and Change Management"

This chapter explains what software configuration management and change management are all about.
- Chapter 14, "Implementation of ISO 9001 Using TeamConnection"

This chapter provides information about using TeamConnection to meet some key ISO 9001 requirements.
- Appendix A, "Component Hierarchy Creation Aid"

This appendix provides a useful component hierarchy creation aid that you can use when planning for the component hierarchy.
- Appendix B, "Build Tree Aid"

This appendix provides a useful build tree aid that you can use when planning for your build trees.

□ Appendix C, "Useful REXX Programs"

This appendix provides some useful REXX programs.

□ Appendix D, "Special Notices"

This appendix describes special notices related to TeamConnection.

□ Appendix E, "Related Publications"

This appendix describes related publications.

FRONT_1.2 Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

1.0 Chapter 1. Application Development Challenges and TeamConnection

Application development teams today face tremendous challenges. While both applications and application development environments are becoming more and more complex, development teams are challenged to continuously increase code quality through adherence to process standards and audit trail management. The application development tools available today to help with these issues often come up short, addressing only specific problems and not providing sufficient tool integration to deliver fully integrated software development environments.

Applications under development are made more complex by the need to support advanced graphical user interfaces (GUIs), object-oriented and client/server technologies, and heterogeneous distributed computing. Client/server applications are moving past decision support into the second generation of mission-critical, high-volume transaction processing, which brings with it additional complexity in application design.

The team's development environment is also becoming more complex in an effort to meet these challenges. Members of the development team are distributed on a local area network (LAN), yet they have the same requirements for coordinated team programming support that is available in the centralized host environment. Object technology, with an increasing focus on support for reuse, and tool integration at the data level are requirements that add to the complexity of the development environment.

Successful software development organizations are measured by Software Engineering Institute (SEI) maturity levels and by compliance with ISO 9000 standards. These maturity levels and standards present new challenges to development teams that are being asked to deliver software products to market at faster and faster rates.

Subtopics

- 1.1 How Can TeamConnection Help?
- 1.2 What Does TeamConnection Do?
- 1.3 TeamConnection Customer Value

1.1 How Can TeamConnection Help?

TeamConnection helps address application development challenges by:

- ☐ Enabling you to organize your componentry for reuse
- ☐ Enabling you to manage the changes to your software more efficiently
- ☐ Helping you to more efficiently rebuild your applications after they have been modified
- ☐ Improving team communication and deployment through e-mail notification on an action required, application changes, and project status
- ☐ Packaging your applications for delivery so you can get them to your customers more quickly and with greater reliability
- ☐ Providing a development model that can help increase your SEI maturity level and improve your application quality through reliable, efficient, and repeatable processes
- ☐ Providing extensive problem tracking and change control for both fine-grained (like for example data elements) and coarse-grained application development objects (files)
- ☐ Providing extensive reporting that can be used for tasks such as impact analysis, project status, project management, quality analysis, and workload balancing
- ☐ Providing a secure repository for your software assets and the information about them, with role-based access
- ☐ Providing an application development information model along with repository services that provide the basis for tool integration
- ☐ Supporting nondisruptive schema evolution of the information model. End users can maintain their investment in current tools and their existing skill base working with those tools and still allow for introduction of new tools into the development environment
- ☐ Supporting data exchange with other platforms and modeling tools, including migration of legacy data, through IBM Exchange for OS/2, an optional feature that will be available in a later release
- ☐ Providing backup and recovery facilities

1.2 What Does TeamConnection Do?

TeamConnection integrates software configuration management services and object-oriented repository services on a semantic model for tool integration to support application development in an OS/2 client/server team programming environment. Applications can be targeted for any platform and can be stand-alone, client/server, or distributed applications.

TeamConnection delivers the function your application development teams need to manage development data, application versions, and application configurations. The integrated problem tracking and change control system ensures that, while your application developers are more productive, your project leaders can effectively manage the development process and track progress. TeamConnection automates and streamlines your application build process and integrates it with the version and change control processes. The build process is tied to release management and extended to provide a framework for delivery of software.

An open, extensible information model provides the vehicle for data sharing between a set of integrated tools using TeamConnection. This object-oriented information model enables an extensible architecture, thus ensuring continued support for new versions of existing tools, as well as new tools that are brought into the active repository environment.

TeamConnection supports application development customers who employ model-driven development to:

- ☐ Support application development by teams charged with the responsibility of developing complex applications quickly and maintaining those applications as responsively modeled objects
- ☐ Facilitate data sharing, where all members of the development team, irrespective of their roles, have a common understanding of an application's data (that is, semantic consistency of data is enforced)
- ☐ Exploit a LAN topology for the application development environment
- ☐ Use distributed application development data, which further exploits a LAN topology in the management of application development resources

TeamConnection is built on an object-oriented database, which allows TeamConnection's repository to store fine-grained application development data. Its software configuration management services work equally well with fine-grained modeled objects and conventional coarse-grained objects (files). The integration of these higher-level services with more traditional repository services such as constraint checking, version management, concurrent and distributed access, and data exchange, make TeamConnection unique in the industry.

TeamConnection provides the following services:

- ☐ Configuration management
- ☐ Release management
- ☐ Version control
- ☐ Integrated build
- ☐ Packaging and distribution support
- ☐ Problem tracking and change control
- ☐ Reporting
- ☐ Backup and recovery
- ☐ Object-oriented repository
- ☐ Information model
- ☐ Data constraints
- ☐ Information model schema evolution

1.3 TeamConnection Customer Value

TeamConnection "marries" repository technology with software configuration management services in a team environment. Thus you can store all of your development data in one place regardless of form and provide a consistent set of services and processes for your entire development team, not just programmers.

TeamConnection helps you manage your development process by organizing your data, controlling changes to it, and providing reporting capability.

TeamConnection increases your team's productivity by automating common development tasks, notifying team members when action is required, and providing a vehicle for tools to share data.

TeamConnection helps improve the quality of your products by providing repeatable, reliable processes that ensure proper authorization for changes.

TeamConnection provides an open, customizable, and scalable environment that can evolve and grow with your development team.

2.0 Chapter 2. Terminology

As a first-time user of TeamConnection, the product's terminology can sometimes be a bit overwhelming. But don't be intimidated by it. For every tool there is a learning curve, some steeper than others. TeamConnection has a fairly steep learning curve, not because it is a difficult-to-learn product, but because it can do so much. So how can we make it easier for you? We start with terminology.

In the sections that follow we explain some of the terminology used in TeamConnection. Figure 1 shows some of the TeamConnection terms and their interrelationships.



Figure 1. TeamConnection Relationship among Objects

Subtopics

- 2.1 Configuration Management
- 2.2 Release Management
- 2.3 Version Control
- 2.4 Integrated Build
- 2.5 Packaging and Distribution Support
- 2.6 Integrated Problem Tracking and Change Control
- 2.7 Report Facility
- 2.8 Backup and Recovery
- 2.9 Repository and Model Support
- 2.10 Information Model
- 2.11 Data Constraints
- 2.12 Information Model Schema Evolution
- 2.13 Application Development Tools Integration

2.1 Configuration Management

Configuration management in TeamConnection provides the ability to identify, organize, manage, and control access to your development data. Within TeamConnection, data is divided into one or more families. A **family** represents a complete and self-contained collection of related **managed objects** (development data, which can be files or objects) and data about the managed objects. Data within a family is completely isolated from data in all other families, even those on the same server. One family cannot share data with another, except through external synchronization. TeamConnection has mechanisms that can be used to implement customer-defined synchronization processes.

The structure used to organize data within a family is called the **component hierarchy** (see Figure 2). Each node within the hierarchy is called a component. A **component** is a TeamConnection object that simplifies project management, organizes project data into structured groups, and controls configuration management properties.



Figure 2. The Family Component Hierarchy

The hierarchy provides a mechanism for organizing components of data into structured groups. Some common ways of grouping data in a component hierarchy are by function, by execution platform, for access control, or for communication needs. The component hierarchy should reflect the organizational requirements of your development team and can be modified over time as those requirements change.

Each managed object under TeamConnection is managed by a component. Components that manage one or more managed objects are usually the leaf components of your hierarchy, while the branch components are used for organization, access control, notification, and problem reporting.

Control of TeamConnection objects and the authority to access TeamConnection data are managed by components and granted to users through component ownership and the access lists associated with the components. Users are given access to objects in a specific family through their TeamConnection user IDs. Each family has at least one **superuser**. The superuser has privileged access to the family and acts as the TeamConnection administrator. All TeamConnection users have some basic capabilities that can be further extended by the superuser or his or her delegates. Ownership of each component is assigned to one user, who is responsible for managing all development data related to that component and the objects it manages. The owner has automatic authority to perform many actions.

Each component has an **access list** (see Figure 3). The access list manages user access (beyond the owner) to objects controlled by the component by assigning users specific authorities. Default authority groups are shipped with TeamConnection and can be tailored to meet your needs. Access authority is inherited by lower-level components.

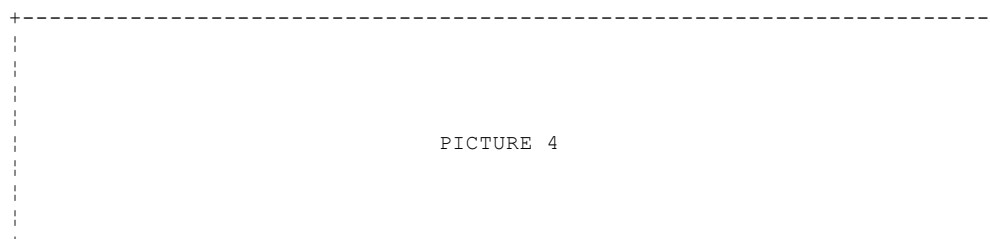


Figure 3. Access List

Each component can also have a **notification list** (see Figure 4). Notification messages are sent to an electronic mailing address that is specified when the user's TeamConnection ID is created. Some notification is automatic. For example, the owner of a TeamConnection object is always notified when actions are performed on that object. Users receive notification when an action affects their user ID or requires them to

perform an action in return. Additional notifications can be configured through the notification list, which maps user IDs to interest levels. Like access authority, notification is inherited by lower-level components.



Figure 4. Notification List

A **managed object** is development data that is stored by TeamConnection and retrieved by name (see Figure 5). A managed object can also be thought of as a TeamConnection part. The managed objects controlled by TeamConnection can be text objects, binary objects, modeled objects, or empty. They are always owned by a component.

Text objects are ASCII files, which are generally created by an editor. Examples are source code, test cases, documentation, README files, and install files. Binary objects are files that are generally created by a tool. Examples are OBJ files that a compiler produces, EXE files that a linker produces, and other files that a tool produces which have a format understood only by that tool. Modeled objects are defined in the information model. They are created and used by a tool but not stored as a file. In this case, the tool uses a set of TeamConnection application programming interfaces (APIs) to store and retrieve modeled objects in the underlying repository. The command line is not used for modeled objects. All of the capabilities associated with files are equally applicable to objects. Examples of modeled objects are shared data elements and VisualGen logic records. Empty objects are placeholders, for example, for the output of a future build.



Figure 5. Managed Objects

When you create a TeamConnection managed object, you are placing an existing development object under TeamConnection control. To change a managed object, you check out the object from TeamConnection, make changes, and then check in the changed object, thus updating the TeamConnection server. TeamConnection stores additional information about the managed object (such as who changed it and why) each time an action is performed against it. This information can be queried at any time.

Common actions against managed objects include:

- Create** To store an object from your workstation in the TeamConnection server, or to store an object from a tool in the TeamConnection server, or to create an object as a placeholder
- Check out** To get a copy of the object onto your workstation for update
- Check in** To put an updated object back in TeamConnection
- Extract** To get a copy of an object onto your workstation without any intention of making changes to it
- Build** To execute one or more build steps, such as compile, link, or generate

Tools such as VisualGen and DataAtlas might hide these details from the end user by invoking the actions on behalf of the end user.

2.2 Release Management

A **release** is a logical organization, or mapping, of all managed objects that are related to an application. A release does not affect the physical location of a managed object; instead it provides a logical view of the managed objects that must be built, tested, and distributed together (see Figure 6). Each release is created against a managing component, which provides the access and notification mechanisms. However, the managed objects in a release can be managed by many components. Also, a specific object can be in multiple releases. Each time a development cycle begins for the next version of an application, a separate release can be defined. Each subsequent release of an application references many of the same managed objects.



Figure 6. Release Management

The release manages whether or not managed object changes are tied to defects and features. If the **track** process (described in "Planning Your Release Processes" in topic 3.7.2) is turned on, object changes require a defect or feature number. If the track process is turned off, managed object changes can be made without a defect or feature number. Because you can configure your change process by release, you can have a low level of control early in the development cycle, add controls as the release comes closer to shipment, and use the highest level of control after the release has been shipped to customers and is being maintained.

2.3 Version Control

Versioning is the making of copies of data at some meaningful point in order to return to that point at a later date, if necessary (see Figure 7). Most programmers version their code, even if they are not using a software configuration management system. For example, Kim is a programmer who has just added a nice little routine to an application and validated that the new routine works. Before writing the next routine for the application, Kim makes a copy of the code that works. If subsequent changes break the application, she can always go back to the copy of the application that worked.

TeamConnection versions application development activity at a release level. As compared with tools that record and track versions of file changes, the TeamConnection release-level versioning model maintains "snapshots" of the release in addition to snapshots of managed objects. A **version** of a release is the set of the correct versions of all of the managed objects that make up a release. As changes are committed to the release, the release is **frozen**, or saved, so that each evolution can be retrieved.

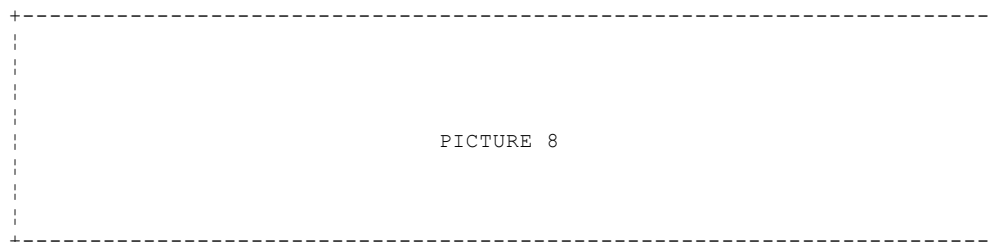


Figure 7. Versioning

The TeamConnection versioning model not only enables you to work on the correct versions of individual managed objects, but it also maintains and presents those managed object versions within the context of the identified version of the release. You can change between work activities, such as problem resolution on an old release and enhancements to the current release, yet the complete version of each release is re-created to reflect the correct versions of its included managed objects.

The versioning model is implemented through the use of **work areas** (see Figure 8). To update a release, you create a work area, which is a "sandbox" where you can update managed objects and do builds without affecting the release. The work area you create maps to the most current version of the release. You **check out** a managed object from the work area, edit it, and then **check in** the managed object to the work area. Your changes now exist in the TeamConnection server but are visible only in the work area. After building in the work area and testing, you update the release by committing the work area. On a **commit**, the release is frozen and then updated with the changes from the work area. This provides the history of changes and the ability to re-create each evolution of the release.

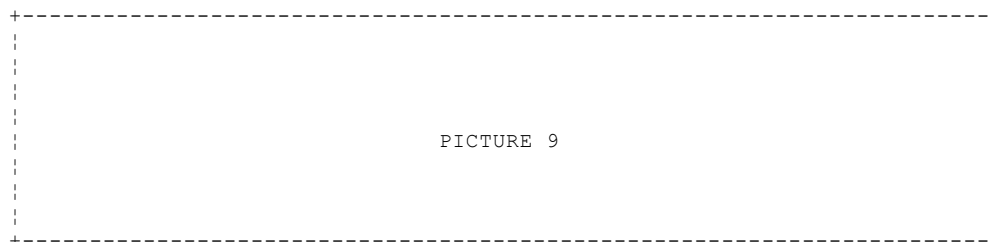


Figure 8. Work Area

A version of a release is the set of the correct versions of all managed objects for a complete release. Maintaining copies of all managed objects in a release for each version would require exorbitant disk space. That is why versions of a release are actually mappings to the correct versions of the included managed objects. In addition, versions of the text files are stored as reverse delta versions to conserve space.

TeamConnection also provides optional support for concurrent development, so you can update managed objects in parallel with other developers. Its graphical merge tool helps you resolve conflicting changes in text objects.

2.4 Integrated Build

TeamConnection's integrated build function automates and optimizes the process of building individual managed objects, entire releases, or multiple releases (see Figure 9). Building can encompass many different processes such as compiling, linking, generating, document processing, binding an application to a database, or invoking a command file. With distributed build, managed objects or releases can also be built for the enterprise client/server environment, that is, for target platforms different from the server platform. Integrated build functions are available for OS/2 and MVS. In the future, additional platforms will be supported.



Figure 9. Integrated Build

Integrating the build capability within TeamConnection provides value that standard make functions cannot match.

☐ **Repeatable and reliable**

Once the build structure is set up, TeamConnection can provide a repeatable, reliable build. The same output is produced from the same input when you use the same set of translation or build rules. You can also build different output from the same input by using different translation rules, for example, to compile for a different platform. You can create the build structure, using an object-based GUI or the command line interface.

☐ **Minimizes resources**

TeamConnection detects out-of-date conditions in a build structure and rebuilds only those parts of the release that have to be rebuilt, thus minimizing the resources required to accomplish the build. Because of the way that time stamps are kept in a LAN environment, out-of-date conditions cannot always be detected in a file system. However, because TeamConnection stores all input, its build function does not rely on file system time stamps, thus improving its performance over make functions.

☐ **Builds multiplatform applications**

Client/server applications require that code for multiple platforms be kept in sync. The build function in TeamConnection can incorporate the build procedures for multiple platforms and synchronize the build of the related applications.

☐ **Distributes build across available machines**

TeamConnection can manage a pool of available machines to build an application. It can determine which parts of a build have to be serial and which can be parallel. Then it can parcel out the work to available machines. For example, a pool of OS/2 machines might be defined for compiling C code. At night, programmer machines can be added to this pool for large build processes. With this capability, a multistep build is no longer constrained by the speed of a single machine.

☐ **Builds objects and files**

TeamConnection provides software configuration management services on a broad range of objects (files and objects), all stored in a single object-oriented repository. With these capabilities, TeamConnection can support third-generation language (3GL), fourth-generation average (4GL), model-driven, and object-oriented development paradigms. Objects for these paradigms can be stored together in TeamConnection. Where applications are written in a mixture of 3GL, 4GL, and object-oriented, TeamConnection can build them together and keep them in sync. For example, a VisualGen application might work with an application written in COBOL. TeamConnection can store and manage both applications within the same build structure.

☐ **Creates one job**

Given the scope of the build, all required processing is handled in one job. Thus integrity is ensured.

Subtopics

2.4.1 Build Mechanisms

2.4.2 Distributed Build

2.4.1 Build Mechanisms

A **build tree** is a structure that graphically defines how the managed objects are built together (see Figure 10). The build tree is a complete description of the dependencies that managed objects have on one another and of all the steps required to build the release. Each executable step in the build tree is a **build event**. For example, the compilation of C source into object code is a build event.

A **builder** is a TeamConnection object that describes how to perform a build event: how to translate input managed objects into output managed objects. For example, one builder might know how to transform C++ source code into object code. A different builder might know how to transform object code into an executable. The builder identifies the environment where the transform will execute, passes parameters to and from its build script, defines the basic rules of successful completion, and invokes a build script.

The **build script** that the builder uses is essentially a binding between TeamConnection and a transform tool. The build script invokes the tool. Because of the way it uses build scripts to invoke tools, TeamConnection is highly extensible and can be used with a variety of build tools, including:

- ☐ Application generators, such as VisualGen
- ☐ 3GL compilers
- ☐ Linkers
- ☐ Object-oriented C++ compilers
- ☐ Preprocessors such as for DB2 and CICS
- ☐ Document processors

In OS/2, a build script is usually a command file. It can be as simple as a single string that invokes the tool. In MVS, it is a job control language (JCL) fragment. The JCL job stream can do anything that a normal JCL job stream does. For example, a build script designed to move an application into test might copy the application executable into a test loadlib along with a set of test cases.

A **parser** is a tool that knows how to read a source file and report back a list of its dependencies. A parser simplifies the job of defining a build tree. It frees you from having to know which dependencies a managed object has on other managed objects. For example, a C parser knows how to read a C source code file and report back a list of files included by the source file. If dependencies are found during the parse that are not reflected in the build tree, the build tree is automatically updated. A set of sample language parsers and build scripts is provided with TeamConnection.

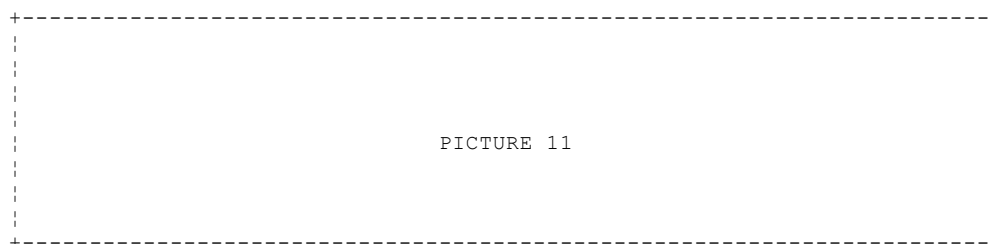


Figure 10. Build Tree, Build Event, Builder, Build Script, and Parser

Once the build tree, builders, build scripts, and parsers are set up, TeamConnection has a reliable, repeatable build process that minimizes the time and resource required to build an application. When a build is requested, the build process determines out-of-date managed objects (objects that have been changed or whose inputs have been changed) within the build tree, parses any out-of-date objects to find any new dependencies, and uses the builders and build scripts to rebuild only the out-of-date objects in the requested environment.

2.4.2 Distributed Build

In TeamConnection, the build processor is where build tools such as compilers, linkers, and generators execute. In the case of compilers and linkers, this is typically the target environment where the application will run.

In the build process described in the previous section, the build actually consists of putting the **build scope** on the **job queue** (see Figure 11). The build scope is the collection of build events that must occur to complete the build. The job queue is the list of build scopes to be completed.

The **build processor** is the TeamConnection process that actually invokes the tools, such as compilers and linkers, that construct an application. The build processor also manages a cache in order to reduce file transfer overhead. Because the build processor may be physically located on a separate CPU from the data, each build processor is paired with a build agent, which typically runs on the TeamConnection server machine. The **build agent** acts on behalf of the build processor to query the job queue for work and retrieve managed objects when the build processor needs them.

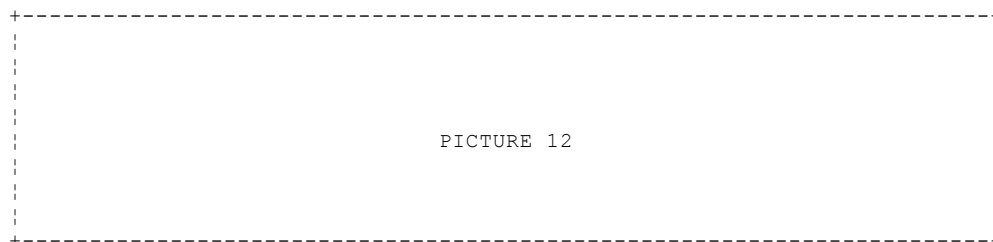


Figure 11. Distributed Build

After the administrator starts a build processor and its corresponding build agent, the build agent polls the job queue for build events. TeamConnection understands which build events within the job queue have to be performed in a specific order. TeamConnection also understands the environment required for a build event, such as MVS or OS/2. These mechanisms enable parts of a build to occur in parallel, if multiple build processors are available. They also enable builds to be distributed to the appropriate target environment for execution.

With this design TeamConnection can handle a wide variety of topologies such as:

- ☐ Building on a remote machine
- ☐ Building a distributed application where application parts execute on different machines
- ☐ Building an application that can run on multiple machines (same source code built for both OS/2 and MVS)

As each build event is completed, the TeamConnection server is updated with the return code and any output, such as the compiled object code. After the build is completed, the TeamConnection client that requested it is notified.

This architecture provides considerable flexibility. For example, an MVS programmer could edit, compile, and debug an application on an OS/2 workstation, using IBM VisualAge for COBOL for OS/2. When the application is ready for system test or production, the programmer requests a build for MVS. The build is executed on MVS, and the executable is automatically brought back to TeamConnection so that it can be versioned along with the source.

2.5 Packaging and Distribution Support

Packaging and distribution support provide a bridge from development to the production environment. In TeamConnection, packaging is any of the steps necessary to distribute TeamConnection-managed applications to software users or onto the machines where software is to be used. To do this, TeamConnection extends the build process to include transformation of executable and nonexecutable files into a distribution-ready form.

Your process might require more than one packaging step to prepare software for distribution. Examples are (see Figure 12):

- ☐ Tersing (compressing) files
- ☐ Packing files together
- ☐ Encrypting files
- ☐ Organizing and grouping files
- ☐ Adding prerequisite files and programs such as installation utilities
- ☐ Updating asset or inventory managers (license tracking)
- ☐ Transferring the files to physical media
- ☐ Distributing the software electronically

Any or all of these steps can be automated by incorporating them into the build tree. Unlike conventional build steps, packaging build steps often do not change the content of the release; they simply organize or package the release in preparation for some form of distribution. In addition the output of packaging build steps generally does not consist of physical objects that are stored back into TeamConnection; instead, the output is abstract or conceptual. So if the build object is up-to-date, it might not have produced something in TeamConnection; instead it might represent the fact that files were transferred to a remote server, for example.

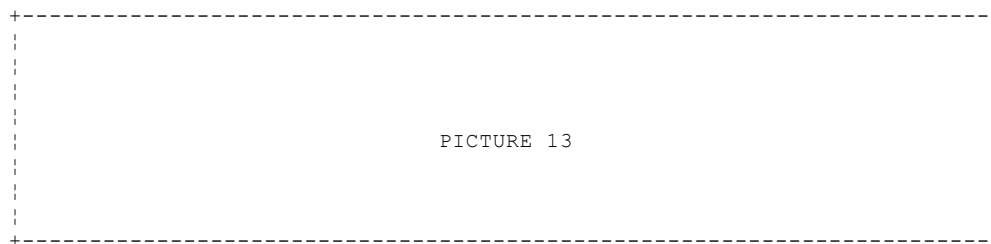


Figure 12. Build Process Packaging Steps

Subtopics

2.5.1 TeamConnection's Packaging Utilities

2.5.2 Your Own Distribution Tool

2.5.1 TeamConnection's Packaging Utilities

In addition to its highly customizable packaging support, TeamConnection focuses on solutions to electronically distribute software. For electronic distribution TeamConnection has two utilities:

- The Gather/2 tool, an automated data mover for server- or file-transfer-based distribution
- The NVBridge/2 tool, which automates the installation and distribution of software or data using IBM NetView Distribution Manager/2 (NVDM/2) as the distribution vehicle.

Together these two utilities provide the ability to move an application in TeamConnection, with its associated files such as installation files and documentation, to an OS/2 subdirectory and then invoke NVDM/2 to catalog the application, distribute it, and install it (see Figure 13).

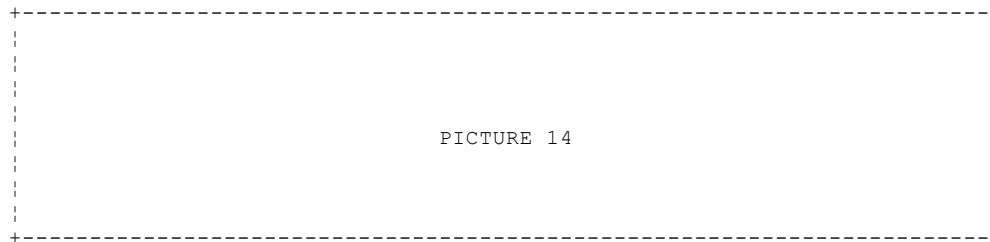


Figure 13. Electronic Distribution

The following scenario demonstrates the value of integrating these tools into TeamConnection: An application managed by TeamConnection is in production when an error is detected. The maintenance team opens a defect (defects are discussed in "Integrated Problem Tracking and Change Control" in topic 2.6) for the error. A work area is created to incorporate the modifications into the current release. After testing, the work area is committed to the release. The release build tree includes the gather and NVDM/2 steps. When these steps are built, the updated application is moved into the file system, and, through NVDM/2, it is distributed and installed. Because the build tree was used, the audit team can be sure that the application includes the fix.

2.5.2 Your Own Distribution Tool

There are many tools for electronic software distribution. You can build a bridge similar to NVBridge/2 to invoke the distribution tool of your choice. To simplify this task, TeamConnection includes the mini-utilities that were used to create NVBridge/2. Although those utilities are unique to NVDM/2, you can use them to develop your own bridge.

2.6 Integrated Problem Tracking and Change Control

Problem tracking (see Figure 14) and change control enables you to manage and control your development process. TeamConnection tracks reported problems and design changes and retains information about the life cycle of each. A **defect** is used to record each reported problem. A **feature** is used to record each proposed design change. The information recorded about defects and features enables you to report on the who, what, when, why, and where of modifications as well as where a particular defect or feature is in the development cycle, and where the release is in the development cycle. For instance you can use the defect and feature information to answer these questions:

- ☐ How many features have been implemented or still have to be implemented?
- ☐ How many defects are open?
- ☐ How many features are still in test? (Do I have to move resources to test?)
- ☐ How many defects have been opened against each managed object? (Where are my code quality problems?)

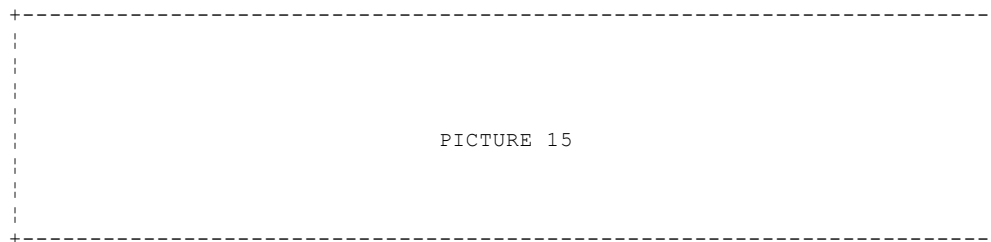


Figure 14. Problem Tracking

Any TeamConnection user can open a defect or feature. Each defect and feature must be opened against a specific component within the component hierarchy. The defect or feature can be reassigned to a more appropriate component within the hierarchy if necessary. The owner of the component to which it is assigned automatically becomes the owner of the defect or feature. This ownership can also be reassigned.

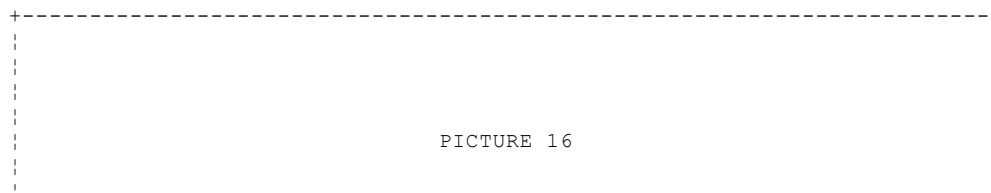
Defects and features go through a configurable process. As they move through the process, team members who have to take action are notified. This automatic notification helps keep your process running smoothly and aids in team communication. It might even reduce the number of meetings that use development cycles for unproductive work.

If you choose to require that changes to managed objects can be made only in association with a defect or a feature, changes can be made only when the defect or feature is in the working state. This enables you to require the correct level of authorization before a change can be made.

Although build objects and versions are release-specific, defects and features are familywide. Thus a specific defect or feature can be implemented for multiple releases. In addition problems fixed in a maintenance release are likewise fixed in the development release, so your customers will not find an old problem in a new release.

TeamConnection ties the process of managing defects and features to the change control process through the use of the track process. If you use the **track** process, changes to managed objects must be tied to defects and features. Because changes are implemented in work areas, with the track process, you can build, test, and ultimately commit to the release individual changes.

If you use the track process, you can also use the **driver** process. The driver process integrates multiple defects and features before committing them to a release by adding their work areas to a driver (see Figure 15). As this is done, changes can be built together and tested together before committing them to a release.



|
+-----+
|

Figure 15. Driver

With the ability to configure processes, you can tailor TeamConnection to best suit your environment. You can adjust the process over time to fit your release's stage in its life cycle.

2.7 Report Facility

The report facility in TeamConnection is a structured query language (SQL) query facility that has been extended for the object-oriented database. End users can form dynamic queries to quickly find objects of specific interest, such as defects assigned to them, objects they have checked out, or all changes associated with a particular feature. The GUI can be used to form queries.

Queries can be stored and used for extensive reporting, auditing, and project management, as the following examples demonstrate:

- ☐ **Impact analysis:** Tell me all releases in which a managed object is used.
- ☐ **Quality analysis:** Tell me how many defects have been reported against each component in a release.
- ☐ **Workload balancing:** Tell me how many defects are being worked. Tell me how many defects are being tested. Use this information to move resources between development and test.
- ☐ **Project status:** For features scheduled in this release, tell me how many are open, implemented, tested, and complete.

Output from queries can be input to other tools such as Visualizer, which will graph the results of a report.

2.8 Backup and Recovery

TeamConnection's backup and recovery facility enables TeamConnection data to be stored on alternate media for archival purposes. Because of data consistency concerns, TeamConnection does not merge old data with new or current data on restoration. See Chapter 11, "Maintaining Your TeamConnection Environment" in topic 11.0 for more details about backup and recovery of the TeamConnection database.

2.9 Repository and Model Support

TeamConnection leverages object-oriented database technology and services for its persistent data store. The need to separate the three levels of repository definition (conceptual, logical, and storage) is rarely challenged. TeamConnection has married this three-tiered "schema" specification with the LAN's physical topology (see Figure 16).

Conceptual View. TeamConnection's information model defines the conceptual schema that is available for all integrated tools. This information model is a fine-grained model with a robust metamodel supporting the following:

- ☐ Inheritance
- ☐ A relationships framework supporting attributes, relationships on relationships, cardinality constraints (minimum and maximum), ordering, and complex controlling semantics
- ☐ Attributes, on both the objects (or entities) and relationships
- ☐ Complex data typing on attributes (for example, structures, pointers)
- ☐ Passive constraints
- ☐ A model framework providing abstract supertype (managed object) services classes from which the semantic classes of the information model can be subtyped. This allows for consistent repository and/or software configuration management services to be applicable for classes as they are introduced into the information model definition.

Logical View. Each tool presents the information that an end user needs to perform the tasks supported by the tool. This subset of information is the logical view, which is defined by the tool and stored in TeamConnection. It can be used by the same tool for other tasks or by other tools that use the same subset of information to satisfy the information needs of their end users. New logical views can be built from existing views.



Figure 16. Information Model Views

The logical view is presented to the tool through an object model API and implemented as a tool cache. This API supports the tool's logical view of the information model for the storage and retrieval of modeled objects from the TeamConnection persistent store. When object instances are made available to the tool, the tool can manipulate (create, delete, update) these instances through the object model APIs (for example, get and/or set attributes, create and/or delete objects). Tools lock or unlock and commit changes for their view. The tool cache exists on the client machine where the tool executes. It manages the tool's view of data in memory and controls the transactions necessary to write changes back to the object-oriented database.

Storage View. The logical views enable TeamConnection to optimize the storage of the information sent through the APIs. This separation also provides the target layer for mapping the TeamConnection services and data schema to the underlying persistent storage database.

2.10 Information Model

The information model provides the common semantic definition by which integrated tools share data and interact (see Figure 17). It is an *objectified* entity-relationship model, in which entities are replaced by more powerful objects. **Semantic classes** represent the traditional view of the information model; they include (but are not restricted to):

- ☐ Entities, attributes, relationships
- ☐ Shared data elements
- ☐ High-level language constructs (for example, COBOL data structures)
- ☐ Relational database definitions
- ☐ Hierarchical database definitions
- ☐ Application visual parts
- ☐ Application systems definitions
- ☐ Business rules
- ☐ Build process definitions (in support of automated build processing)
- ☐ Bulk data (multiple file types are managed)

Through inheritance, these semantic classes are enabled to the services defined on the managed object class to allow for future extensions by IBM, vendors, and users, to attach to these services in a straightforward manner. The semantic classes are shared by the suite of tools working with TeamConnection as an active development-time repository. This shared data schema is the foundation of the data integration seen through the integrated suite of tools and is the integrating platform through which development tools work together.

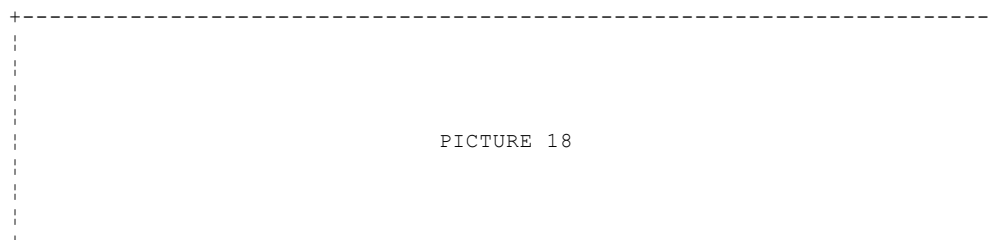


Figure 17. Information Model

The information model supports object definitions that range from fine-grained classes (for example, an entity) to coarse-grained classes (for example, a COBOL source file). This breadth enables TeamConnection to support 3GL, 4GL, model-driven, and object-oriented development paradigms in an integrated fashion, providing a common semantic model and layer of software configuration management services to each.

The scope of the semantics over which these classes act is related to the tool suites enabled to the integration framework. This scope includes support for data descriptions from the analysis and design phase of development through the specific characteristics for given languages or database technology. These semantics provide the definition for the tools in the Team Suite, including VisualAge Requirements Tool, DataAtlas, and VisualGen.

2.11 Data Constraints

Constraints are methods that ensure that the data is consistent with the definition of the information model. They are defined as part of the model definition and are useful in two ways. First, they enforce a global, consistent standard for stored data. Second, tools do not require code to validate data in the model before use. The constraint framework is triggered whenever the tool cache writes updates to the persistent data store.

2.12 Information Model Schema Evolution

As tools are added to the set of tools using the information model for storage of fine-grained information or as the current tools evolve to include additional functions, it will be necessary to add semantics to the current information model. To minimize the effort required in the customer environment, TeamConnection provides schema evolution facilities that enable the information model to be extended without recompiling existing code. These facilities are used with each new release of the information model. Not having to recompile the code for additions to the information model is a very important feature, as tools normally ship just the executable form of their products.

2.13 Application Development Tools Integration

TeamConnection was designed for extensibility and easy integration. The software configuration management architecture enables many IBM and non-IBM tools to be easily used with TeamConnection for storage and control of development data and for building output:

- Tools that transform a set of input into a set of output can be plugged into TeamConnection through the use of a build script.
- Tools can provide transparent access to the software configuration management services by invoking the command line interface.
- TeamConnection function can be extended by invoking tools from TeamConnection user exits. User exits are available before and after each command.

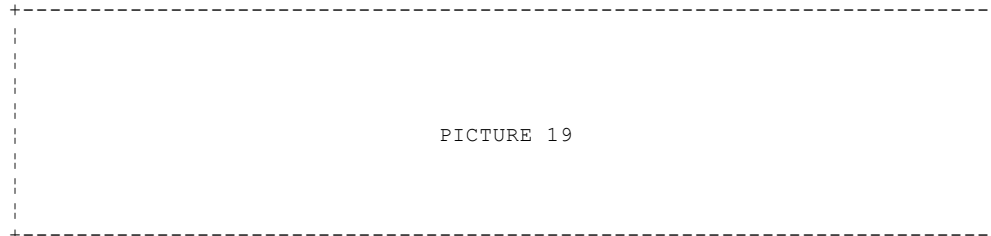


Figure 18. Application Development Tools Integration

The repository architecture includes a set of APIs and an open extensible information model. Tools can share data in the information model, store their unique data in the information model, and use the common software configuration management and repository services. You can customize much of the software configuration management integration in your development environment by using the methods identified above. However, more seamless software configuration management integration and all repository integration must be implemented by tool vendors working with the TeamConnection development team. To facilitate such integration, a TeamConnection Vendor Enabling program is actively pursuing integrations that will deliver high-value solutions to you.

3.0 Chapter 3. Planning for TeamConnection

"Good planning is the key to success."
-- Wise Man --

Some might even say: "Planning is everything." Planning does play an important role in enabling you to get the most out of TeamConnection. In this chapter we offer some general advice on planning for TeamConnection.

Subtopics

- 3.1 Installation Planning
- 3.2 Family Planning
- 3.3 User IDs and Host Lists
- 3.4 The Component Hierarchy
- 3.5 Component Ownership, Access Lists, and Notification Lists
- 3.6 Releases
- 3.7 Planning Your Processes
- 3.8 Planning Your Build Environment
- 3.9 Choosing a Naming Convention

3.1 Installation Planning

This section describes the minimum machine requirements and the program requirements for valid operating environments for TeamConnection for OS/2.

Subtopics

3.1.1 Hardware Requirements

3.1.2 Software Requirements

3.1.1 Hardware Requirements

TeamConnection runs on IBM and IBM-compatible personal computer hardware configurations supported by IBM OS/2 Version 2.11 or higher.

Note: Required memory and DASD can vary according to your installation and configuration choices and user application requirements.

Table 1 lists the hardware requirements for the TeamConnection **server** machine.

Table 1. TeamConnection Server Hardware Requirements		
Type	Description	
Processor	50 MHz 486-based processor or higher	
Monitor	VGA graphics monitor or higher-function (XGA) monitor and its appropriate adapter	
Pointing device	A mouse or other pointing device	
Memory	48 MB memory minimum	
Disk space	Operating system and prerequisites	75 MB
	TeamConnection server code	60 MB
	Swapper space	64 MB
Communications support	Network card supported by TCP/IP for OS/2	
CD-ROM drive	A CD-ROM drive (internal or external) for installation of product	

Table 2 lists the hardware requirements for the **build processor** machine.

Table 2. Build Processor Hardware Requirements		
Type	Description	
Processor	50 MHz 486-based processor or higher	
Monitor	VGA graphics monitor or higher-function (XGA) monitor and its appropriate adapter	
Pointing device	A mouse or other pointing device	
Memory	32 MB memory minimum; more might be necessary, depending on compilers, linkers, and programs	
Disk space	Operating system and prerequisites	75 MB
	TeamConnection build processor code	15 MB
	Swapper space	45 MB
Communications support	Network card supported by TCP/IP for OS/2	
CD-ROM drive	A CD-ROM drive (internal or external) for installation of product	

Table 3 lists the hardware requirements for the **build agent** machine.

Table 3. Build Agent Hardware Requirements		
Type	Description	
Processor	50 MHz 486-based processor or higher	
Monitor	VGA graphics monitor or higher-function (XGA) monitor and its appropriate adapter	
Pointing device	A mouse or other pointing device	
Memory	32 MB memory minimum; more might be necessary, depending on compilers, linkers, and programs	
Disk space	Operating system and prerequisites	75 MB
	TeamConnection build agent code	40 MB
	Swapper space	45 MB
Communications support	Network card supported by TCP/IP for OS/2	
CD-ROM drive	A CD-ROM drive (internal or external) for installation of product	

Table 4 lists the hardware requirements for the **client** machine.

Table 4. TeamConnection Client Hardware Requirements

Type	Description						
Processor	25 MHz 386-based processor or higher						
Monitor	VGA graphics monitor or higher-function (XGA) monitor and its appropriate adapter						
Pointing device	A mouse or other pointing device						
Memory	12 MB minimum						
Disk space	<table> <tr> <td>Operating system and prerequisites</td><td>75 MB</td></tr> <tr> <td>TeamConnection client code</td><td>25 MB</td></tr> <tr> <td>Swapper space</td><td>32 MB</td></tr> </table>	Operating system and prerequisites	75 MB	TeamConnection client code	25 MB	Swapper space	32 MB
Operating system and prerequisites	75 MB						
TeamConnection client code	25 MB						
Swapper space	32 MB						
Communications support	Network card supported by TCP/IP for OS/2						

3.1.2 Software Requirements

Table 5 lists the software requirements for both servers and clients.

Table 5. TeamConnection Clients and Servers Software Requirements	
Type	Description
Operating system	<p>IBM OS/2 Version 2.11 or higher</p> <p>Note: If double-byte character set (DBCS) support is required, use one of the following versions of OS/2:</p> <ul style="list-style-type: none"><input type="checkbox"/> IBM OS/2 J Version 2.11 or later<input type="checkbox"/> IBM OS/2 H Version 2.11 or later<input type="checkbox"/> IBM OS/2 T Version 2.11 or later<input type="checkbox"/> IBM OS/2 P Version 2.11 or later
Communications support	<p>One of the following:</p> <ul style="list-style-type: none"><input type="checkbox"/> TCP/IP 2.0 for OS/2<input type="checkbox"/> Anynet Sockets over SNA 2.0
MVS build	<ul style="list-style-type: none"><input type="checkbox"/> One of the following:<ul style="list-style-type: none">- TCP/IP Version 2.2.1 for MVS or higher- Anynet Feature for VTAM Version 4.2<input type="checkbox"/> C/370 run-time libraries for Version 2.2 or higher
Packaging function	NetView Distribution Manager/2 Version 2.0 or higher when using the NVBridge/2 tool

3.2 Family Planning

As mentioned in "Configuration Management" in topic 2.1, a family represents a complete and self-contained collection of related managed objects and data about the managed objects. The family represents the physical database. Data within a family is totally isolated from data in all other families, even those on the same server. Families cannot share data.

Before the creation of a family and the structure within that family, it is important to consider the following:

- ☐ Whether, and how, to separate project data between families
- ☐ How to arrange the component structure
- ☐ How to organize the releases
- ☐ Which processes to use

Careful planning of the families that an organization will use is an important first step in the planning process. Before deciding whether to have one family or create additional families, consider these guidelines:

- ☐ Data cannot be shared between families, so group all development projects that share source data within the same family.
- ☐ You can create new families if needed. If it is not clear whether there is a need for one or more families, consider starting with one. It is easier to separate one family than it is to combine two families.
- ☐ The more families, the more administrative work.

Why would you want multiple families?

There are a several reasons why a company would want to have multiple families:

- ☐ Development is spread across the globe and there are no high-speed links between the development sites, so using one single family is not realistic.
- ☐ There is a need for separate education and test and production families.
- ☐ It is a customer requirement. Take as an example the case where a software house supports several customers, and developers from the various customers have access to the development data. The customers require that their specific development data be kept separate from other customer's development data.

There is no right or wrong in organizing multiple families. In the case of the software house, it might be advisable to keep generic development data in one family and customer-specific development data in their specific families.

3.3 User IDs and Host Lists

On receipt of a TeamConnection command, the TeamConnection server uses the following lists to authenticate the identity of a TeamConnection client:

User list A list of TeamConnection user IDs with access to a TeamConnection family.

Host list A list of login ID and client host name combinations that may access the TeamConnection server. In OS/2 an entry in the host list consists of a user ID and a host name. The user ID can be specified in the *CONFIG.SYS* file as *SET TC_USER*; the host name must be set in *CONFIG.SYS* as *SET HOSTNAME*.

A TeamConnection user ID identifies a user independently of the system's login ID (as defined in TCP/IP). A single user ID can be associated with multiple login IDs at multiple hosts; the login ID and the host name make up a user's host list. Likewise, a single login ID and host name combination can be associated with multiple users.

Users are given access to the TeamConnection objects in a specific family through their user IDs. Each family has at least one superuser, with privileged access to the family. The superuser gives other users the authority to perform some set of actions on particular objects. Depending on the authority granted to a user, that user might in turn be able to grant some equal or lesser level of authority to other users. However, the ability to grant authority for some actions is reserved for the superuser. The superuser can perform all actions.


3.4 The Component Hierarchy

You can organize your component hierarchy in several ways. For example, as shown in Figure 19, one component hierarchy might mirror the application development organization hierarchy, such as department, section, team, or unit of development; another component hierarchy might reflect the software architecture of the applications under development, such as application, GUI, and database; and yet another might reflect the platform for which you are developing.

When you set up your component hierarchy, consider that all defects and features are recorded by component, and the owner of a component becomes the default owner of the defects and features for that component. This is important because defect and feature owners automatically receive a considerable amount of authority over the defects and features they own.

To create a component, you have to have superuser authority, or you have to have inherited authority. To have inherited authority means that you already are a component owner and that you can create child components under this component. Initially, however, someone with superuser authority, such as the family administrator, creates the first components.

When the superuser has created a component, he or she transfers ownership to the appropriate user. Once ownership has been transferred, the new component owner can create child components and grant access authority to other users.



PICTURE 20

Figure 19. Component Hierarchy Models

Subtopics

- 3.4.1 Reflecting the Product Organization
- 3.4.2 Parallel Components
- 3.4.3 Multiple Parents
- 3.4.4 Component Evolution
- 3.4.5 Naming the Components

3.4.1 Reflecting the Product Organization

If you create your component hierarchy to store software or documentation source files, it is best to reflect the product organization at the top level. You can then create descendant components to reflect the development or maintenance responsibilities.

3.4.2 *Parallel Components*

Your component hierarchy can consist of several parallel hierarchies so that you can easily restrict access to certain related components. For example, if you have vendors working on your development team, you might want to restrict their access to certain information. You can create a parallel hierarchy that contains only the information that they require.

3.4.3 *Multiple Parents*

Components can have more than one parent. A component that has more than one parent inherits authority and notification from both.

3.4.4 *Component Evolution*

Your initial component hierarchy is not necessarily going to be the same as your hierarchy a year from now. It will change as your organization grows and your needs change. Remember that you can change the parents of a component as well as delete or rename the component.

When you plan your component hierarchy, it is helpful to first sketch it on paper. You can then use this sketch to make a table in which you note information about each component, such as the type of parts you want to control with the component, which releases a component will manage, and which processes each component and release will initially follow.

3.4.5 Naming the Components

During the planning stage, decide on a component naming convention. Do you want each component name to reflect the type of data it is managing? If so, you have to understand the content, function, execution platform, or other characteristics of the parts the component will manage. For example, the name of a component that manages data for the GUI of your application might begin with the characters *GUI*. Do you want component names to be in all lowercase characters, all uppercase characters, or in mixed case? The database is case sensitive. Therefore, when you are consistent with your component names, your users will have less difficulty finding objects in the database. The names you use must be unique within the family. Other TeamConnection users will be able to create components, so you should publicize your naming convention so that everyone can adhere to it.

TeamConnection users have to access TeamConnection data, and they will have difficulty finding it if they do not understand and follow your naming convention.

3.5 Component Ownership, Access Lists, and Notification Lists

Each component in the hierarchy has an owner. Initially that owner is the person who creates the component. After the component is created, the owner can, at any time, transfer ownership to another person.

Ownership of a component is critical. A component owner has authority to perform a wide variety of actions on that component and the parts contained in that component, as well as on all its descendant components and their parts. For example, the owner has authority to give other users access to the component and its parts, delete the component, and destroy parts within that component.

You might create many of the initial components for your development organization, but you probably will not want to remain the owner of them all. As you are planning your component hierarchy, determine the owner of each component. The owner of the root component, the component at the top of the hierarchy, should be the person with overall responsibility for the project. If several other people have responsibility for various pieces of the development project, those people should own the descendant components that relate to their part of the project.

The owner of a parent component has the same level of authority for all its descendant components.

Subtopics

3.5.1 Access Lists

3.5.2 Notification Lists

3.5.1 Access Lists

Access authority to TeamConnection objects is managed by the components defined in TeamConnection. Each component has an access list that controls access to development objects. Access authority is inherited by lower-level components. When a user has authority to perform actions within one component, that authority is inherited for all descendant components. A user whose user ID appears in the component's access list either has authority to perform any action or is restricted from performing any action listed in the specified authority group.

As soon as a TeamConnection user ID and a host list entry are created for a user, that user automatically has the authority to perform the following basic actions within the family's component structure:

- ☐ Open defects and features
- ☐ Modify the information for the user's own ID
- ☐ Display information about any user ID
- ☐ Add notes to existing defects and features
- ☐ Search for information within TeamConnection to create reports

TeamConnection users get authority to perform additional actions when they own a TeamConnection object, or when authority is explicitly given to them by the component owners. Because this authority is inherited, care has to be taken when assigning component ownership and when granting access authority.

The authority to perform various TeamConnection actions is based on three types of authority levels:

Implicit authority Every TeamConnection object, such as a component, file, or defect, has an owner. The object owner automatically receives authority to perform certain actions.

Explicit authority Some users need additional authority to perform actions against objects that they do not own.

Superuser authority A user with TeamConnection superuser authority can perform any TeamConnection action.

3.5.2 Notification Lists

On request, TeamConnection notifies users when certain actions are performed on certain objects. Notification messages are sent to the electronic mailing address specified when the user ID is created.

Some notification is automatic. For example, the owner of a TeamConnection object is always notified when actions are performed on that object. Additionally, users receive notification when an action affects their user IDs or requires them to perform an action in return. For example, a user receives notification when his or her user ID is added to an access list.

Users can receive additional notification. For example, a manager might want to be notified whenever a defect is opened against a component. The component owner can explicitly request that TeamConnection send notification to the manager.

Each component has a *notification list* that controls who is notified of specified TeamConnection actions. Notification is inherited by lower-level components. When a user is to be notified that a specific action occurred within one component, that user is also notified when that action occurs in any of the descending components. Subsets of actions can be placed into interest groups. Each interest group is a group of actions of which a user with a certain role wants to be notified. It is comparable to authority groups. For example, a developer might want to be notified when defects are opened or closed, whereas the lead developer must be notified not only when defects are opened or closed but also when defects are sized or verified. Interest groups can be created and modified by the family administrator.

In addition to *role-oriented* interest groups, three general interest groups exist: *high* interest, *medium* interest, and *low* interest. Thus users who do not necessarily have a TeamConnection role can be notified according to their *level* of interest.

3.6 Releases

A release is a user-defined TeamConnection object that contains all objects that must be built, tested, and distributed as a single entity.

Each TeamConnection part is managed by a component. An entire application is likely to contain parts from more than one component. Because you probably want to use the same parts in more than one version of an application, TeamConnection also groups parts into releases. A release is a way of identifying the exact version of all parts that comprise an application at a certain point in time.

Each part in TeamConnection is managed by at least one component and contained in at least one release. One release can contain parts from many components; a part can be included in several releases.

Each time a development cycle begins for the next version of an application, you can define a separate release. Each subsequent release of an application references many of the same parts as its predecessor. However, each release links to a particular version of individual parts. Thus maintenance of an older release can occur at the same time as development of a newer release (see Figure 20).

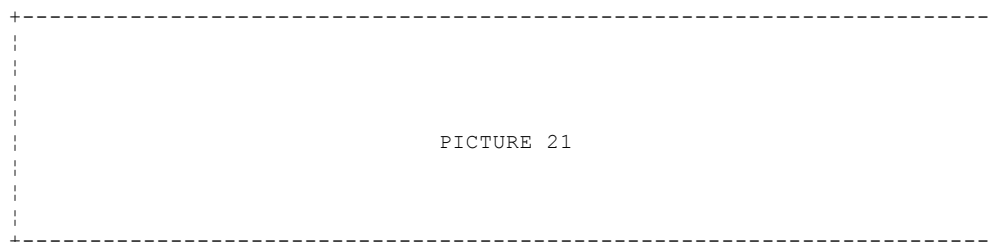


Figure 20. Releases

Every release is associated with a component that manages which users can access the parts in the release and which users are notified when certain actions occur.

Subtopics

3.6.1 Selecting Serial or Concurrent Development

3.6.2 Naming Your Releases

3.6.3 Using Remote Databases

3.6.1 Selecting Serial or Concurrent Development

The release can be set up in serial development or concurrent development mode. In serial development a part is locked when a user checks it out, and no one else can update the part as long as it is checked out. In concurrent development more than one user can simultaneously have the same part checked out. When TeamConnection detects that someone else has made changes to a part that you are checking in, it notifies you that a collision has occurred. You can then reconcile the changes, using the TeamConnection merge program.

You specify the mode in which your users will work when you create the release. Be aware, however, that you cannot change the mode after you have set it.

3.6.2 Naming Your Releases

Decide how you are going to name the releases you create. You might want to name your releases according to the product or object you are building. For example, *prod1r1* for release 1.1 of your application, or *using1r1* for the book files for release 1.1 of your application. To make it easier on your users, continue using the basic naming convention used for your components. The names you use must be unique within the family.

3.6.3 *Using Remote Databases*

Each administrator can define the topology for each family. The meta data objects, such as users, defects, features, and host lists, reside in the family database. However, the contents of each part for each release can, at the administrator's discretion, reside in a separate database. You might want to use this topology to split heavily used releases across several storage units for better performance.

You can specify a separate database for the part data when the release is created. Use the *db* flag in the release command or the **Database** field on the Create Releases window to specify the database you want the release to use.

3.7 Planning Your Processes

Before you create your family's components and releases, decide which processes you are going to use initially during development (see Figure 21).



Figure 21. Processes

A TeamConnection process enforces a specific level of control of components and releases. TeamConnection is shipped with a set of predefined processes for both components and releases, so you can provide different processes for each to follow. You can use these processes, or you can configure your own processes, using the predefined subprocesses.

You probably already have a process for tracking problems as well as a process for tracking suggested improvements to your applications. If you want to continue to use those processes, determine how you can best group the TeamConnection subprocesses to reflect your current process. If you do not have an existing method, decide how tightly you want to control part changes and track defects and features.

Figure 22 shows the various states that different TeamConnection objects can go through according to the process that is followed. Study this information before you determine how to use TeamConnection processes.

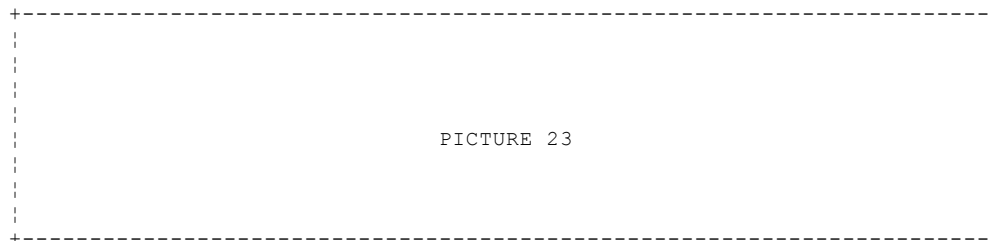


Figure 22. States

Subtopics

3.7.1 Planning the Component Process

3.7.2 Planning Your Release Processes

3.7.1 Planning the Component Process

A component's process (see Figure 23) determines how much planning and designing are required before work on a defect or feature begins and whether the originator is required to verify that the work was done correctly.

When choosing a process for a component to follow, consider the type of data within the component. For example, the parts within one component might contain complex code that is time-consuming to fix. Before any defects or features are accepted, the work has to be designed and sized, using the *preship* process. Parts within another component contain code that is relatively easy to fix and test. The defects and features for this component do not have to be designed and sized, so the *prototype* process, which contains no subprocesses, is followed.

For components, you can require users to follow any, all, or none of the following predefined subprocesses:

dsrDefect	Design, size, and review fixes to be made for defects.
dsrFeature	Design, size, and review changes to be made for features.
verifyDefect	Verify that the fixes work.
verifyFeature	Verify that the features have been implemented correctly.

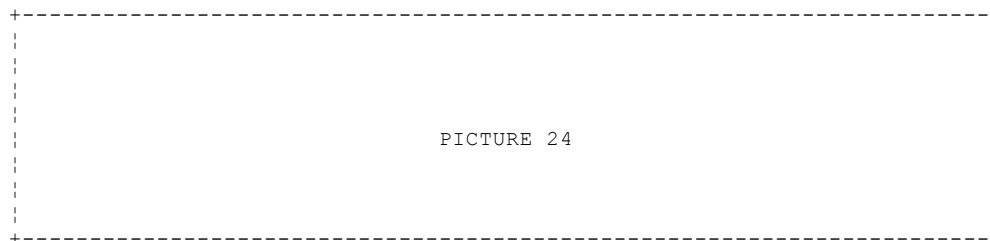


Figure 23. Component Processes

3.7.2 Planning Your Release Processes

A release's process determines to what extent part changes are tracked and the steps for integrating changed parts into a build. Release processes control the day-to-day work involved in fixing defects and implementing features, as well as building the product. The type of process control you want to enforce on a release may change over time.

For releases, you can require any, all, or none of the following predefined subprocesses (see Figure 24):

- track** This subprocess is TeamConnection's way of relating all part changes to a specific defect or feature and a specific release. Each work area gathers all parts modified for the specified defect or feature in one release and records the status of the defect or feature. The work area moves through successive states during its life cycle. The TeamConnection actions that you can perform against a work area depend on its current state.

You must use the track subprocess to use any of the other release subprocesses.
- approval** This subprocess ensures that a designated approver agrees with the decision to incorporate changes into a particular release and electronically signs a record. As soon as approval is given, the changes can be made.
- fix** This subprocess ensures that as users check in parts associated with a work area, an action is taken to indicate that they have completed their portion. When everyone finishes, the owner of the fix record (usually the component owner) can mark the fix record as complete. The parts are then ready for integration.
- driver** A driver is a collection of all work areas to be integrated with each other and with the unchanged parts in the release at a particular time. The driver subprocess enables you to include these changes incrementally so that their impact can be evaluated and verified before additional changes are incorporated. Each work area included in a driver is called a *driver member*.
- test** The test subprocess guarantees that testing occurs before the fix is verified as correct within the release.

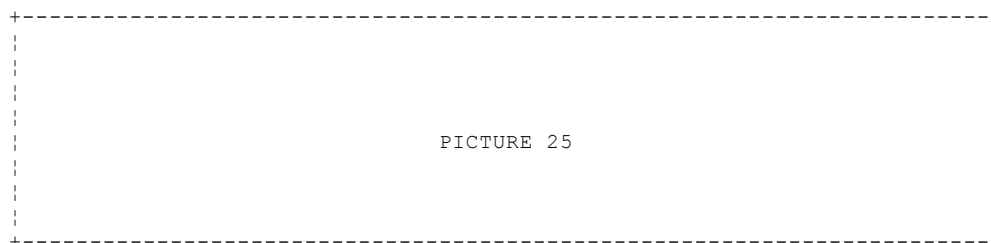


Figure 24. Release Processes

Figure 24 lists the release processes supplied by IBM. Each process combines different sets of TeamConnection subprocesses. A *check mark* under the TeamConnection subprocess indicates that the corresponding process includes it.

Users should understand the meaning of each process and the type of control it enforces. For example, if a stringent release process such as *track_full* is selected, actions must occur in a precise order. Compare this with the *no_track* process where users can freely check parts in and out of TeamConnection.

3.8 Planning Your Build Environment

A TeamConnection build is capable of dealing with:

Heterogeneous applications

A TeamConnection build can handle 3GL, 4GL, and C++ (hybrid) for example. A TeamConnection build can handle anything that can be stored by TeamConnection and needs a *translation*. It is possible to define aggregates of output so that a product consisting of deliverables such as executables on MVS, dynamic link libraries (DLLs) for the LAN server, BookMaster books, and information presentation facility (IPF) files for documentation can be built in one run.

Heterogeneous environments

TeamConnection supports both LAN-based and MVS-based builders. Thus applications that are LAN-based, MVS-based, or based on a combination of both environments can be built. If more environments are needed, writing build agents and servers for the environments that have to be supported should be enough.

Planning the build environment is very important, and it is wise to use a **build administrator**, who would be responsible for planning and implementing the following:

Parsers The parsers provided with TeamConnection can be changed for the specific environment. It is also possible to define your own parser (for example, use the Make make file and load the dependencies through APIs into the database). The parsers are provided on an *as-is* basis.

Builders Defining builders is a one-time job per type of translation. Decisions can be made to split multiple translator steps or combine them (for example, combine precompile, compile, and link steps together in one Build script or separate them). The build scripts are provided on an *as-is* basis. A build step is not necessarily a "translate"; one might consider copying the resulting load module to a library (on MVS) or invoking Gather/2 and NVBridge/2 functionality as build steps (on OS/2).

Topology One can have different topologies for the build servers for distributed build (different platforms, for example, MVS and OS/2). Parallel builds are also possible. If there are lots of compiles to perform, the work can be split and performed on different machines. Builders can be bundled into *pools*, which are serviced by *agents* dispatching and controlling the work within a given *pool*. Obviously the project administrator has to decide on the overall *topology* of TeamConnection components.

3.9 Choosing a Naming Convention

Choosing a naming convention is probably one of the most important tasks in the planning phase. You have to have unique names within the family for your components and releases. Within the same release, you also have to have unique names for parts, work areas, parsers, and builders. You might even want to apply some kind of naming standard for your user IDs (for example, if you want to be able to distinguish subcontractors or the departments to which a developer belongs).

There are several ways to apply a naming convention. If your company or organization already has some kind of naming convention in effect, you can apply it to the names in the family, but you have to make sure that you achieve name uniqueness. One way is to use some kind of prefix or suffix that identifies such objects as components and releases by name and indicates where they belong.

When you create a part in TeamConnection, you can specify only the **base name**, such as **ibmoupd.cob**, or you can specify the directory path in addition to the base name, such as **\Collector\mvs\ibmoupd.cob**. Specifying the **path name** as part of the name lets you have several identical base name parts included in the release--for example:

```
\Collector\mvs\ibmoupd.cob
```

and

```
\Collector\aix\ibmoupd.cob
```

You can also have identical part names within the context of a release as long as their part types are different, such as *file* and *vgdata*. Table 6 shows the length of various field names in TeamConnection.

Table 6. TeamConnection Field Name Lengths		
Field Name	Length	Description
compName	63	Component name
releaseName	31	Release name
workareaName	15	Work area name
baseName	127	Part base name
pathName	195	Path name
userLogin	31	User login name
userName	63	User's full name
userArea	31	User's area or department
authorityName	31	Access authority group name
interestName	31	Interest group name
defectName	31	Defect name
featureName	31	Feature name
ownerName	63	Defect and/or feature owner name
driverName	31	Driver name
builderName	63	Builder name
environment	15	Build environment
parserName	63	Parser name

Appendix A, "Component Hierarchy Creation Aid" in topic A.0 and Appendix B, "Build Tree Aid" in topic B.0 provide some tables to aid you in the creation of component hierarchies and build trees.

4.0 Chapter 4. Installing TeamConnection

In this chapter we explain how to:

- ☐ Prepare to install
- ☐ Install the TeamConnection components
- ☐ Set the environment variables
- ☐ Configure and start the ObjectStore server
- ☐ Verify the installation of TeamConnection
- ☐ Start the TeamConnection client

The chapter is an excerpt from the *TeamConnection Install Guide* that comes with TeamConnection.

Subtopics

- 4.1 Preparing to Install
- 4.2 Installing the TeamConnection Components
- 4.3 Setting the Environment Variables
- 4.4 Configuring and Starting the ObjectStore Server
- 4.5 Verifying the Installation of TeamConnection
- 4.6 Starting the TeamConnection Client

4.1 Preparing to Install

Before you install the TeamConnection code, you must have at least TCP/IP Version 2.0 for OS/2 installed on your workstation. TeamConnection uses the value of the TCP/IP **USER** environment variable to create the initial user in the database. If this variable is not set, installation will fail. Type `set user` at an OS/2 prompt to display the value of the **USER** environment variable. If the variable is not set, or if you want to change its value, type `set user=userid` in your `CONFIG.SYS` file, where `userid` is the ID you want TeamConnection to use. Shut down the workstation and then restart it to accept the changes made to the `CONFIG.SYS` file.

After TCP/IP is installed, update your TCP/IP services and hosts files by following these steps:

1. Update the services file, which is located in the directory where TCP/IP is installed. To determine the directory name, type `echo %etc%` at an OS/2 prompt. Include the family name and port address of the TeamConnection server. The port address can be any four-digit number, as long as it does not already exist in your services file. You might want to ask your TCP/IP administrator to assign you a number. Type the following entries in your services file:

```
# TeamConnection servers
testfam    nnnn/tcp      # port address for the TeamConnection test family
bldsock    nnnn/tcp      # port address for the build server
```

Notes:

- a. For this installation, replace `nnnn` in the above example with appropriate port addresses.
 - b. Follow each line with a carriage return.
2. Update the TCP/IP hosts file. If you have a hosts file, it is located in the directory where TCP/IP is installed. To determine the directory name, type `echo %etc%` at an OS/2 prompt. If the hosts file does not exist, you must configure it through TCP/IP.

Add the following:

- ☐ IP address
- ☐ Server name
- ☐ Alias name of the TeamConnection family server, which is your family name. For this initial installation of TeamConnection, the family name is `testfam`.
- ☐ Alias name for the build socket. For this initial installation of TeamConnection, use `bldsock`.

The following is an example of the entry you would type in your hosts file:

```
9.12.345.67    teamserv.company.com    testfam    bldsock
```

Note: Follow the line with a carriage return.

3. Do the following to verify that you can connect to your TeamConnection family:
 - a. At the OS/2 prompt, type `ping testfam`, where `testfam` is the family name.
 - b. Press `Ctrl+C` to end the command.

If you receive information that is similar to the following, you can successfully connect to your TeamConnection family:

```
PING teamserv.company.com: 56 data bytes
64 bytes from 1.23.457.78: icmp_seg:0. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:1. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:2. time=0. ms
```

If you receive the message "unknown host testfam," you cannot connect to the family. Verify that the data you entered in the hosts and services files is correct, and then try the command again. If you still do not get the correct response, contact your TCP/IP administrator to solve the problem.

Do not install the TeamConnection components until the command successfully completes.

4.2 Installing the TeamConnection Components

TeamConnection provides an installation program that does the following:

- ☐ Installs the family server code
- ☐ Installs the client code
- ☐ Installs a CID client
- ☐ Installs repository code
- ☐ Installs a build processor
- ☐ Installs a build agent
- ☐ Installs an MVS build processor
- ☐ Installs the online documentation
- ☐ Updates your CONFIG.SYS file

This section provides instructions for installing all of the TeamConnection components. If you do not follow these instructions as written, the verification step on page 14 might fail. The time required for installing this program varies depending on your workstation. In general, installation of the product takes about 15 minutes.

Subtopics

- 4.2.1 STEP 1: Starting the Installation
- 4.2.2 STEP 2: Selecting Installation Options
- 4.2.3 STEP 3: Selecting the Components for Installation
- 4.2.4 STEP 4: Completing the Installation

4.2.1 STEP 1: Starting the Installation

1. Insert the CD-ROM into the reader.
2. Type "x:\install" at an OS/2 prompt, where x is the CD-ROM drive letter. The TeamConnection Installation window appears, followed by the Installation Instructions window.
3. Select **CONTINUE**. The Install window appears.

TeamConnection at Large
STEP 2: Selecting Installation Options

4.2.2 STEP 2: Selecting Installation Options

1. From the Install window, indicate that you want to have your CONFIG.SYS file updated. This is the default. If you want existing installation files overwritten during the installation, select **OVERWRITE FILES**. If you select not to have your CONFIG.SYS file updated automatically, you will have to update the file manually after the installation program completes. TeamConnection creates a file called *config.add* that contains your entire CONFIG.SYS file with environment variables and changes to the PATH and LIBPATH statements.
2. Select **OK** or press Enter. The Install - directories window appears.

TeamConnection at Large
STEP 3: Selecting the Components for Installation

4.2.3 STEP 3: Selecting the Components for Installation

Use the Install-directories window to select which components you want to install and where you want them installed.

1. Select **SELECT ALL** to install all TeamConnection components.
2. To verify that the default drive has sufficient space, or to see which other drives are available, select **DISK SPACE**. If you want to use a different drive, do the following from the Disk space window:
 - a. Select the drive that you want to use.
 - b. Select **CHANGE DIRECTORIES TO SELECTED DRIVE**.
 - c. Select **OK** or press Enter. The Install - directories window appears.
3. Select **INSTALL** from the **Install - directories** window to start the installation. The **Install - progress** window shows you the progress of the installation.

4.2.4 STEP 4: Completing the Installation

1. After the installation completes, the Installation and Maintenance window appears with an indication that TeamConnection successfully installed. Select **OK** or press Enter to return to the TeamConnection Installation window.
2. Select EXIT or press F3.
3. Verify that the installation program created a folder called *TEAMCONNECTION GROUP*, which includes several program icons.
4. If TeamConnection automatically updated your CONFIG.SYS file during installation, restart your system now. Go to "Configuring and Starting the ObjectStore Server" in topic 4.4 to continue with the installation process.
If you have to manually add the environment variables to your CONFIG.SYS file, go to "Setting the Environment Variables" in topic 4.3.

4.3 Setting the Environment Variables

If you did not select to have the installation program update your CONFIG.SYS file with the TeamConnection environment variables, TeamConnection created a file called config.add. Update your CONFIG.SYS file with these environment variables. We recommend that you use the default values for this initial installation. If you change the values, you will have to provide them when you set up the TeamConnection server in "Verifying the Installation of TeamConnection" in topic 4.5.

```
OS_ROOTDIR=x:\teamc
TC_FAMILY=testfam
TC_USER=user
TC_BECOME=user
NLSPATH=x:\teamc\bin\%N
TC_DBPATH=x:\teamc\testfam
OS_NETWORK=o4netnp,o4nettcp
OS_TEMPDIR=x:\teamc\temp
LIBPATH=x:\teamc\dll
PATH=x:\teamc\bin
```

Where:

- ☐ **x** is the drive on which the TeamConnection server is installed.
- ☐ **user** is the **USER** environment variable you specified when TCP/IP was installed (see "Preparing to Install" in topic 4.1).
- ☐ **teamc** is the default directory created for the server.

4.4 Configuring and Starting the ObjectStore Server

The TeamConnection server communicates with the ObjectStore database through an ObjectStore server. The database was installed with the family server.

Note: You can have only one version of the ObjectStore database on the server machine.

Do the following to configure the ObjectStore server:

1. Start the ObjectStore for OS/2 Setup program to initialize the ObjectStore server parameters. Start the program in one of two ways:

- ☐ From the TeamConnection Group folder on the OS/2 desktop, select the **OBJECTSTORE SETUP VERSION 4.0** icon.
- ☐ From the directory where the ObjectStore server is installed (teamc\bin is the default), type "ossetup" and press Enter.

The ObjectStore for OS/2 Setup window appears.

2. Select **AUTO START SERVER** if it is not already selected.
3. Select **INITIALIZE SERVER**. This push button is named REINITIALIZE SERVER if you are reinstalling ObjectStore. The Server Initialization window appears. If ObjectStore does not initialize properly, verify that the OS_NETWORK environment variable in your CONFIG.SYS file correctly reflects your client's network configuration.
4. Change the default log file's name, drive, or directory if appropriate.
5. Select OK.
6. If you are reinstalling ObjectStore, the Confirm window appears. Select **YES**.
7. The ObjectStore for OS/2 Setup window appears. Select EXIT.
8. Start the ObjectStore server in one of two ways:
 - ☐ From the TeamConnection Group folder on the OS/2 desktop, select the **OBJECTSTORE SERVER** icon.
 - ☐ From the directory where the ObjectStore server is installed (teamc\bin is the default), type the following command and press Enter.

start /min osserver

Verify that the ObjectStore server (OSSERVER) is active in the OS/2 Window List. To do this, press Ctrl+Esc.

If you experience problems configuring and starting the ObjectStore server, the following files can provide you additional information about ObjectStore errors:

- ☐ OSSERVER.TXT
- ☐ OSC3.OUT
- ☐ OSS.OUT

These files are located in the teamc\temp directory.

4.5 Verifying the Installation of TeamConnection

Follow the steps below to verify that TeamConnection installed successfully.

Subtopics

4.5.1 STEP 1: Create Your Test Family

4.5.2 STEP 2: Start the Family Server

4.5.3 STEP 3: Verify TeamConnection Installation

4.5.1 STEP 1: Create Your Test Family

Before you can verify that TeamConnection is installed correctly, you must create your family. When you create your family, you are loading default information shipped by IBM and creating a superuser ID. A superuser ID is required so that at least one person has privileged access to the family to perform special tasks, such as creating other user IDs. The superuser ID is created using the current value of the TC_USER environment variable and the current host server name. Type the following commands from an OS/2 prompt if you want to see the values that will be used:

```
set tc_user
hostname
```

Before you create your test family, verify that the ObjectStore server (OSSERVER) is active in the OS/2 Window List. To create your family, select the CREATE TESTFAM FAMILY icon from the TeamConnection Group folder on the OS/2 desktop.

4.5.2 STEP 2: Start the Family Server

You can start the TeamConnection server in one of two ways:

- ☐ From the TeamConnection Group folder on the desktop, double-click on the FAMILY SERVER icon.
- ☐ From an OS/2 prompt, type:

```
x:\teamc\bin\teamcd testfam
```

Where:

- x is the drive where the server is installed.
- teamc\bin is the default directory where the server is installed.
- testfam is the name of your TeamConnection test family.

4.5.3 STEP 3: Verify TeamConnection Installation

To verify that TeamConnection installed correctly, do the following:

1. Type the following at an OS/2 prompt in the directory where TeamConnection is installed (x:\teamc\bin is the default), and then press Enter:

```
cd x\teamc\bin
_
univos2
```

Type yes when you are asked if you want to start the build server. When you are asked to enter the build server socket name, type *bldsock*. This is the build socket name that is specified in the hosts file. A file called univos2.log is created.

2. Review univos2.log for a completion code of zero, which indicates that TeamConnection was installed and configured correctly.

4.6 Starting the TeamConnection Client

To start the TeamConnection client, select the TEAMCONNECTION CLIENT icon from the TeamConnection Group folder on the desktop. The Task List window appears.

You now have your initial family installed and running. You or someone else in your organization can use this family to verify that TeamConnection is working properly. You can also use this family to explore and learn about TeamConnection.


```

"A family is a logical unit of related development data."
-- TeamConnection definition of a family --

```

The first thing you do after having installed TeamConnection is to create a family. When you want to create a family, you can use either the `dbcreate` command (for example, by modifying the *Create Testfam Family* settings) or the TeamConnection Family Administrator tool. Using the `dbcreate` command requires that you have set the `TC_DBPATH`, `TC_FAMILY`, `TC_BECOME`, and `TC_USER` environment variables to their appropriate values. If you use the TeamConnection Family Administrator tool, these environment variable values are automatically set for you.

Click on the **TeamConnection Family Administrator** icon PICTURE 26 in the TeamConnection Group folder to start the TeamConnection Family Administrator tool and open the **TeamConnection Family Administrator** window (see Figure 25).

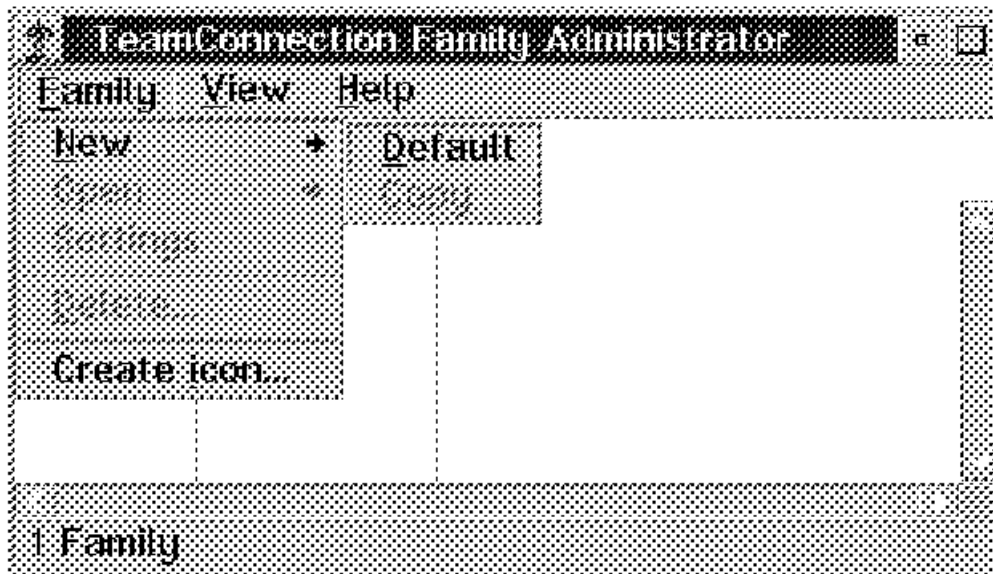


Figure 25. The TeamConnection Family Administrator Window

From the **TeamConnection Family Administrator** window you can access the following *administration* windows and pages:

- ☐ Family Information Page
- ☐ Authority Groups Page
- ☐ Interest Groups Page
- ☐ Configurable Fields Page
- ☐ Component Processes Page
- ☐ Release Processes Page
- ☐ User Exits Page
- ☐ Family Servers Window

The Family Administrator enables you to create and delete a family, change the settings for a family (using any of the *notebook* pages described later in this chapter), create an icon for an existing family, or start the TeamConnection server for a particular family. We recommend that you use the TeamConnection Family Administrator when you want to create or maintain a family.

Subtopics

- 5.1 Family Information Page
- 5.2 Authority Groups Page
- 5.3 The Interest Groups Page
- 5.4 Configurable Fields Page
- 5.5 Component Processes Page
- 5.6 Release Processes Page
- 5.7 User Exits Page
- 5.8 Family Servers Window

5.1 Family Information Page

Use the **Family Information** page to identify the basic family information and the initial superuser for the family.

Subtopics

5.1.1 Fields

5.1.1 Fields

The information you define on the **Family Information** page (see Figure 26) is:

Name

The name of the family you are creating

Path

The fully qualified path name of the directory where you want the family database stored

Port

The TCP/IP port of the family server

Mailer

The mail routine TeamConnection uses to notify users of various events and actions. Refer to the *IBM TeamConnection for OS/2 User's Guide*, SC34-4499, for additional information regarding the mail routine.

Initial Superuser

This information pertains to the initial superuser (typically the family administrator)

Login The login for the initial superuser, for example, *leif*

Name The name of the initial superuser, for example, *Leif Trulsson*

Host The client host information for the initial superuser, for example, *calluna.almaden.ibm.com*

Untitled - Settings

Family Information

Name: fam2008

Path: g:\

Port: 7468

Mailer: mailexit

Initial Superuser

Login: leif

Name: Leif Trulsson

Host: calluna.almaden.ibm.com

Undo Help

Family Information

Authority Groups

Interest Groups

Configurable Fields

Component Processes

Release Processes

User Exits

Create Cancel

Figure 26. Family Information Page

5.2 Authority Groups Page

Use the **Authority Groups** page to set the authority for each authority group. When you set the authority for each authority group, you identify the actions that the various authority levels are allowed to perform. This page can also be used to create new authority groups.

Subtopics

5.2.1 Fields

5.2.2 Push Buttons

5.2.1 Fields

On the **Authority Groups** page (see Figure 27 in topic 5.2.2), you define the following information:

Authority Groups

A list of the currently defined authority groups. Select the authority group you want to modify.

Has these actions

A list of TeamConnection actions. Selecting actions in this list associates them with the authority group you have selected. Likewise, deselecting them removes the association.

5.2.2 Push Buttons

The following push buttons are defined:

Select all

Selects all of the items in the list

Deselect all

Deselects all of the items in the list

Overview

Displays a window listing only those actions this group has authority to perform

New

Creates a new group

Rename

Renames the selected group

Delete

Deletes the selected group

Undo

Removes any current changes and returns the fields to their previous values

Help

Displays help information about the window



Figure 27. Authority Groups Page

5.3 *The Interest Groups Page*

Use the **Interest Groups** page to associate such actions as new, rename, and delete with interest groups.

Subtopics

5.3.1 Fields

5.3.2 Push Buttons

5.3.1 Fields

On the **Interest Groups** page (see Figure 28 in topic 5.3.2), you define the following information:

Interest Groups

A list of the currently defined interest groups

Has these actions

A list of actions that you can assign to the selected interest group. Selecting actions in this list associates them with the interest group you have selected. Likewise, deselecting them removes the association.

5.3.2 Push Buttons

The following push buttons are defined:

Select all

Selects all of the items in the list

Deselect all

Deselects all of the items in the list

Overview

Displays a window listing only those actions that are associated with the selected interest group

New

Creates a new group

Rename

Renames the selected group

Delete

Deletes the selected group

Undo

Removes any current changes and returns the fields to their previous values

Help

Displays help information about the window

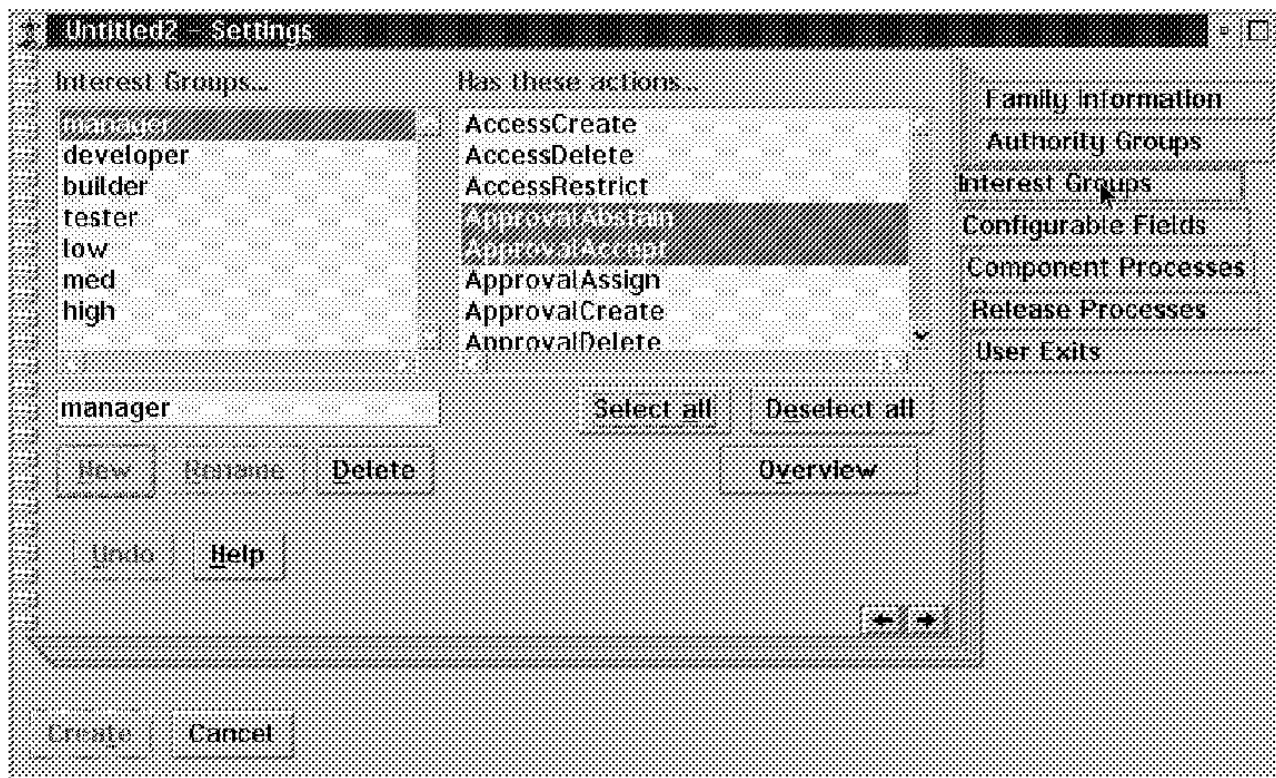


Figure 28. Interest Groups Page

5.4 Configurable Fields Page

Use the **Configurable Fields** page to set up and define the configurable fields for the family. You can add or update configurable fields in the following tables:

- ☐ Defect
- ☐ Feature
- ☐ Part
- ☐ User

The following conditions apply to configurable fields:

- ☐ User and part objects are allowed up to 20 configurable fields each. Defect and feature objects can only have a combined total of 20 configurable fields.
- ☐ Configurable fields cannot be deleted; however, they can be renamed and redefined.
- ☐ Fields for defect, feature and user objects are effective only on create, open, and modify actions.
- ☐ Fields for part objects are effective only on modify actions.
- ☐ A configurable field can contain up to 85 characters. It cannot contain blank spaces or tabs. Configurable fields can contain only character data.
- ☐ The TeamConnection process logic does not use data from configurable fields. You can search the database using configurable fields and you can include configurable fields in reports. For example, if you have a field called *PubImpact*, TeamConnection cannot change the state of a defect based on the value of this field, but users can sort all defects and features by whether or not they impact the publications.
- ☐ TeamConnection help does not reflect changes made to the configurable fields.

Subtopics

5.4.1 Fields

5.4.2 Push Buttons

5.4.1 Fields

On the **Configurable Fields** page (see Figure 29 in topic 5.4.2), you define the following information:

Field types

Identifies the types of configurable fields that are defined for your family. You specify one of these types when you configure a new field. You can create new types, and you can configure the acceptable values for each type. You must have at least one value for each type. The type field can have up to 15 characters, but it cannot contain blank spaces or tabs. The following configurable field types are shipped by IBM:

severity

An indication of how severe the defect is

phase

The current stage in development when the defect was discovered or introduced into the code

symptom

An indication of the problem

defectPrefix

A prefix indicating the type of defect. The prefix attribute of a defect or feature can be used to identify a problem as either a defect or a feature when looking at information regarding both. Use unique prefixes for defects.

featurePrefix

A prefix indicating the type of feature. The prefix attribute of a defect or feature can identify a problem as either a defect or a feature when a user looks at information regarding both. Use unique prefixes for features.

answerReturn

The answer to the defect or feature that the defect or feature owner uses when returning a defect or feature to the originator

answerAccept

The answer to the defect or feature that the defect or feature owner uses when accepting to work on a defect or consider a feature

drivertype

The type of level in which the defect or feature resolution should be included

priority

An indication of the timing or scheduling requirements for resolving a defect or feature

Note: If the default configurable fields shipped by IBM are not installed, the configuration types of priority, phase, and symptom are not used.

Possible Values

This field represents the choices the user has within the configurable field. You can add choices to the default fields shipped by IBM and to the fields created specifically for your family. The value can have up to 15 characters but cannot contain spaces or tabs.

Note: Because a user can abbreviate these values from the command line, you cannot define a value that can be an abbreviation of another value of the same type. For example, you cannot add a value of build to the phase type because a value of building already exists. Also, if a value of 1 exists for the severity type, you cannot add a severity value of 12.

Description

This field contains the description of each name value. The description field cannot contain more than 63 characters, but it can be set to blank. The description appears on the user window interface along with the defined values.

Default

Select this check box to indicate that the defined name is used

TeamConnection at Large Fields

as the default when the user does not enter a value for the configuration type. The config table shipped by IBM does not have any of the values set as defaults.

V1 and V2

Keep these fields set to 0. They are reserved for future use.

5.4.2 Push Buttons

The following push buttons are defined:

New	Creates a new configurable field
Rename	Renames the selected configurable field
Delete	Deletes the existing configurable field
Undo	Removes any current changes and returns the fields to their previous values
Help	Displays help information about the window

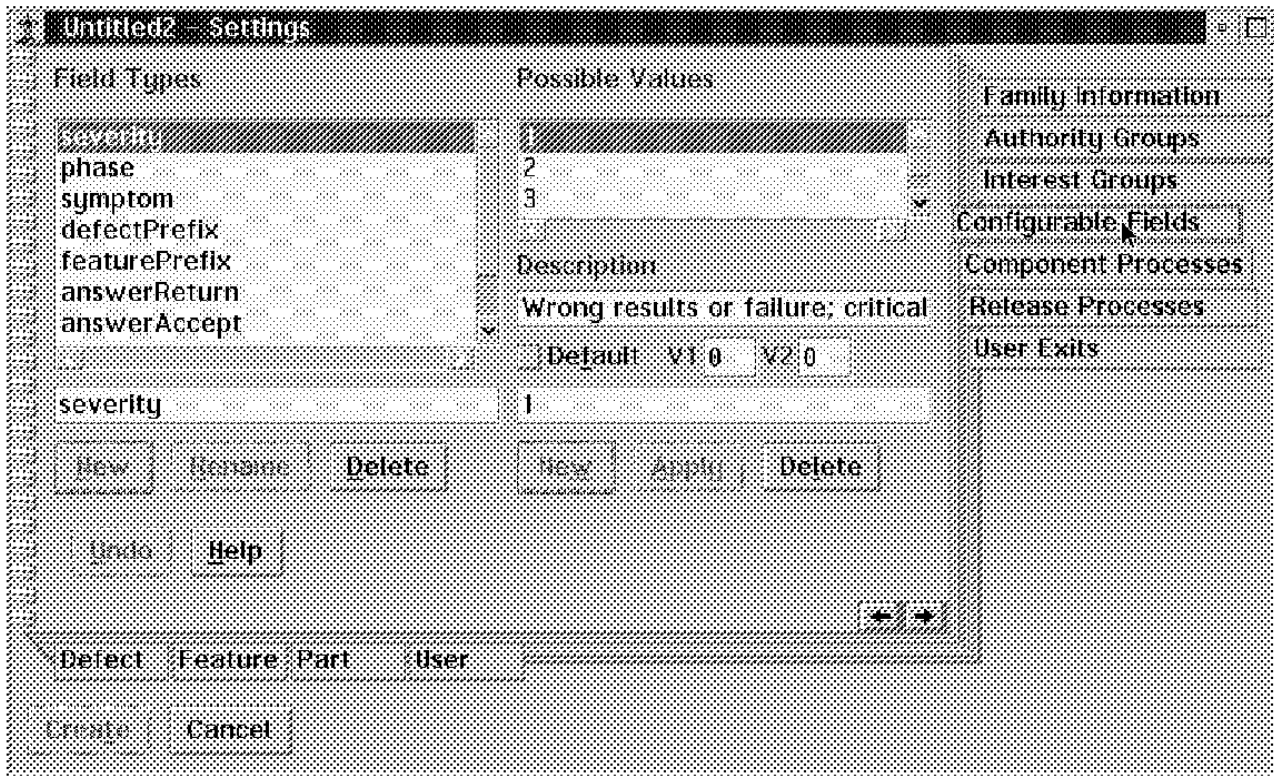


Figure 29. Configurable Fields Page

5.5 Component Processes Page

A component process determines the amount of planning and design required before work begins on a defect or feature written against the component. This is also where you specify if the originator is required to verify that the work was done correctly. TeamConnection is shipped with several predefined processes for components. If these processes do not meet the needs of your development organization, you can create your own processes by combining some of the predefined subprocesses that IBM provides.

Subtopics

5.5.1 Fields

5.5.2 Push Buttons

5.5.1 Fields

On the **Component Processes** page (see Figure 30 in topic 5.5.2), you define the following information:

Component Processes...

A list of the currently defined component processes. Select the process you want to modify.

Has these subprocesses...

A list of subprocesses that you can assign to the selected component process. Selecting subprocesses in this list associates them with the component process you have selected. Likewise, deselecting them removes the association.

5.5.2 Push Buttons

The following push buttons are defined:

Select all

Selects all of the items in the list

Deselect all

Deselects all of the items in the list

Overview

Displays a window listing only those subprocesses that are associated with the selected component

New

Creates a new process

Rename

Renames the selected process

Delete

Deletes the selected process

Undo

Removes any current changes and returns the fields to their previous values

Help

Displays help information about the window

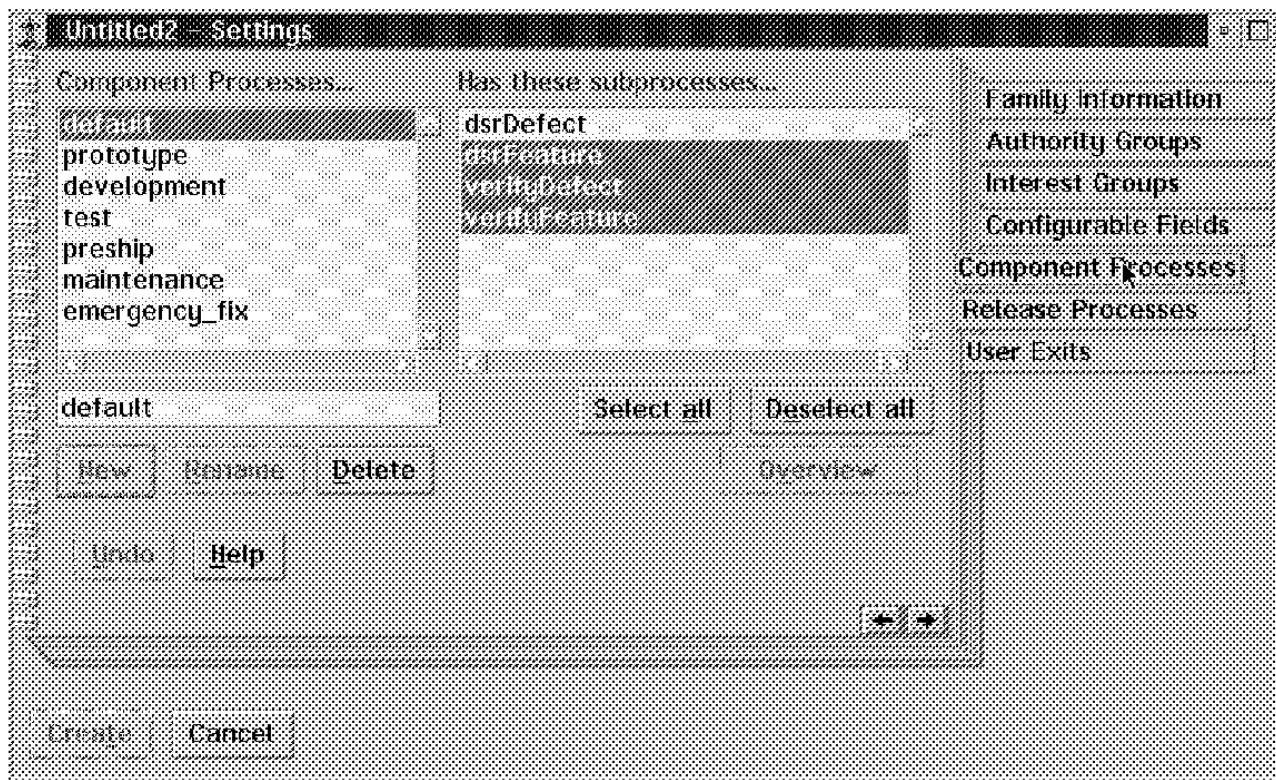


Figure 30. Component Processes Page

For additional information on processes, see "Planning Your Processes" in topic 3.7.

5.6 Release Processes Page

Release processes determine the amount of tracking applied to part changes and the procedure for integrating those changes into build. They control the work involved in developing the parts, fixing defects, implementing features, and building the product. TeamConnection is shipped with several predefined processes for releases. If these processes do not meet the needs of your development organization, you can create your own processes by combining some of the predefined subprocesses that IBM provides.

Subtopics

5.6.1 Fields

5.6.2 Push Buttons

5.6.1 Fields

On the **Release Processes** page (see Figure 31 in topic 5.6.2), you define the following information:

Release Processes

A list of the currently defined release processes

Has these subprocesses...

A list of subprocesses that you can assign to the selected release process. Selecting subprocesses in this list associates them with the release process you have selected. Likewise, deselecting them removes the association.

5.6.2 Push Buttons

The following push buttons are defined:

Select all

Selects all of the items in the list

Deselect all

Deselects all of the items in the list

Overview

Displays a window listing only those subprocesses that are associated with the selected release

New

Creates a new process

Rename

Renames the selected process

Delete

Deletes the selected process

Undo

Removes any current changes and returns the fields to their previous values

Help

Displays help information about the window

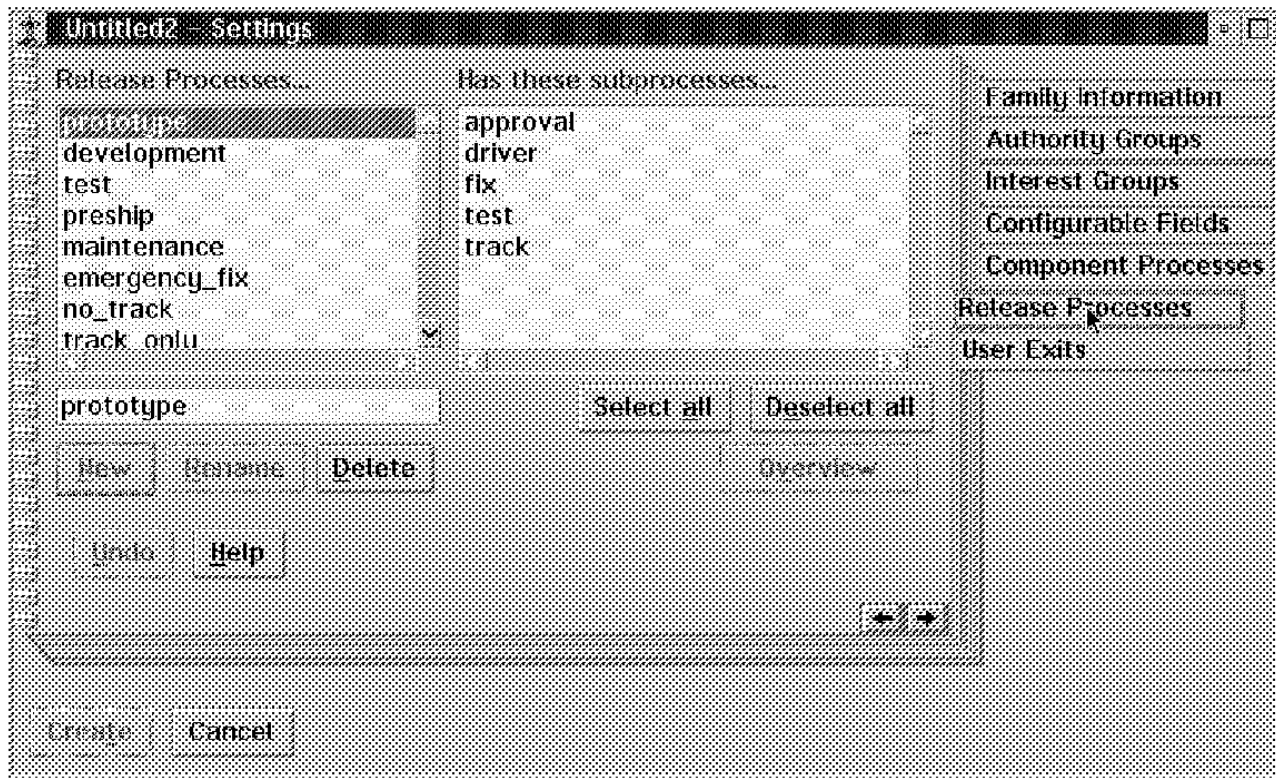


Figure 31. Release Processes Page

For additional information on processes, see "Planning Your Processes" in topic 3.7.

5.7 User Exits Page

Use the **User Exits** page to identify programs that should be executed when certain actions are performed. With user exits, you can specify additional actions that you want performed before completing or proceeding with a specific TeamConnection command action. A user exit enables the TeamConnection server to call a user-defined program during the processing of TeamConnection actions. The program can be an executable file or a command file. Therefore, you can use TeamConnection as a trigger to start non-TeamConnection processing. You can also use user exits to restrict certain TeamConnection actions based on external considerations. For example, you might have a user exit scan C source files to ensure that the source code conforms to the standards defined by your development process. For additional information about writing user exits, please refer to the *IBM TeamConnection for OS/2 User's Guide*, SC34-4499.

Subtopics

5.7.1 Fields

5.7.2 Push Buttons

5.7.1 Fields

On the User Exits page (see Figure 32 in topic 5.7.2), you define the following information:

Actions

The list of existing actions

Programs to run...

The names of the programs to run for the selected action.

before Action checking

The name of the program that starts at the beginning of the TeamConnection action, before any initialization or access checking takes place

before Action processing

The name of the program that starts after all TeamConnection checks are made and TeamConnection is ready to process the command

after Action completes

The name of the program that starts after the TeamConnection action is completed. At this point, the action has been submitted to TeamConnection, and all database or library updates have been committed.

if action or program fails

The name of the program that starts when either of the first two types of user exits are not successful, or when the TeamConnection action is not successful. This user exit program cleans up what the other user exit programs started.

5.7.2 Push Buttons

The following push buttons are defined:

- Overview** Shows an overview of the user exits and stages for the selected actions
- Undo** Removes any current changes and returns the fields to their previous values
- Help** Displays help information about the window

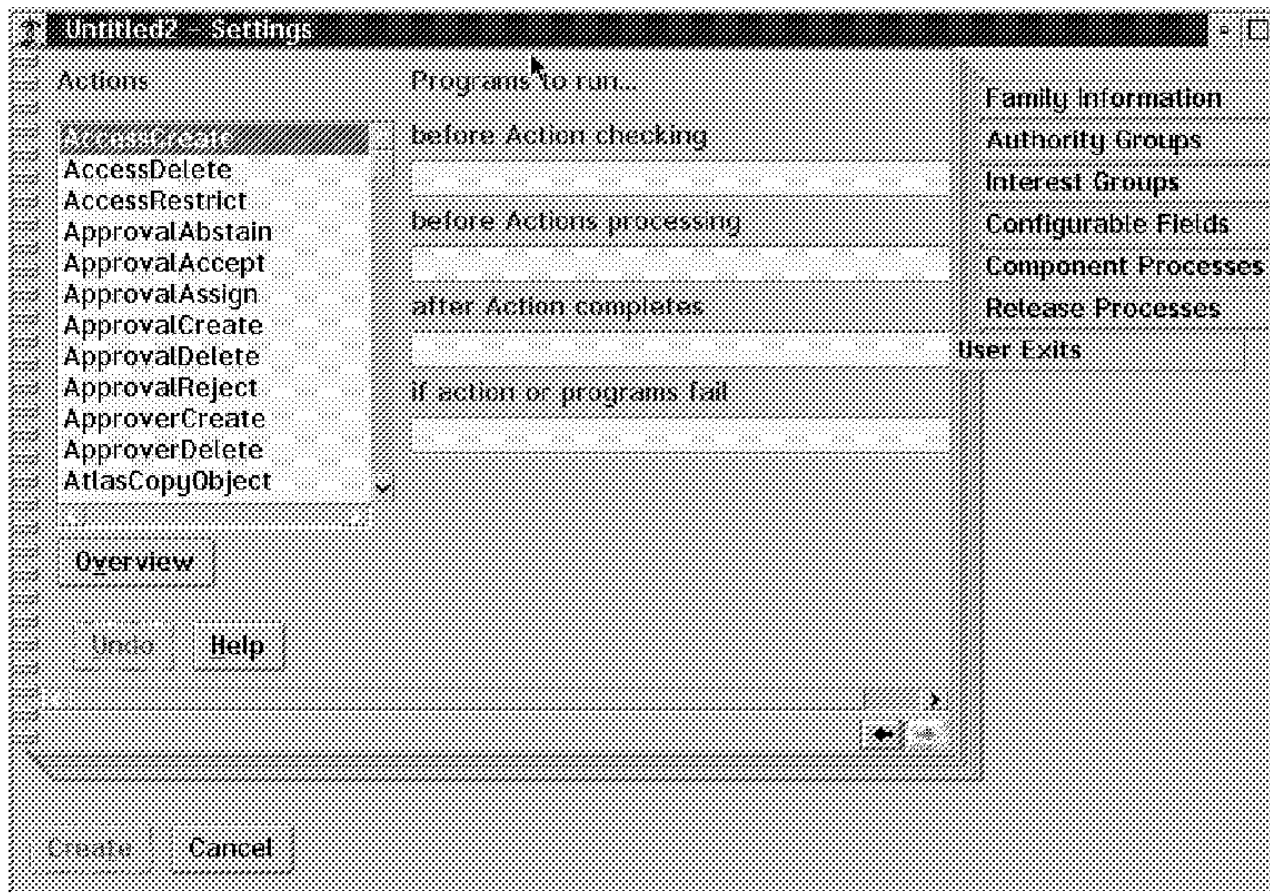


Figure 32. User Exits Page

See the *IBM TeamConnection for OS/2 User's Guide*, SC34-4499, for more information about user exit routines.

5.8 Family Servers Window

Use the **Family Servers** window to start or stop family server daemons and the notification server for a family (see Figure 33).

Start/Stop Both

Select the Start/Stop Both button to start or stop both servers. If the servers are not running, this button is labeled **Start Both**. If the servers are running, this button is labeled **Stop Both**.

Help

Displays help information about the window

Family Server

The following fields and push buttons are defined:

Start/Stop

Select the **Start/Stop** button to start or stop the server. If the server is not running, this button is labeled **Start**. If the server is running, this button is labeled **Stop**.

Daemons

The number of daemons that are to be started. A daemon is a process that runs as a background task and provides access to the TeamConnection database. Valid values are 0-255.

Maintenance Mode

Locks the family into a read-only mode. This mode is useful when you are making changes to the family. It prevents others from changing the contents of the database before the completion of your administrator tasks.

Save

Saves the contents of the information area to a file. The information area is a log of the commands and messages that TeamConnection issues when starting or stopping the server.

Clear

Clears the log field

Notification Server

The following fields and push buttons are defined:

Start/Stop

Select the **Start/Stop** button to start or stop the notification server. If the server is not running, this button is labeled **Start**. If the server is running, this button is labeled **Stop**.

Save

Saves the contents of the information area to a file. The information area is a log of the commands and messages that TeamConnection issues when starting or stopping the server.

Clear

Clears the log field

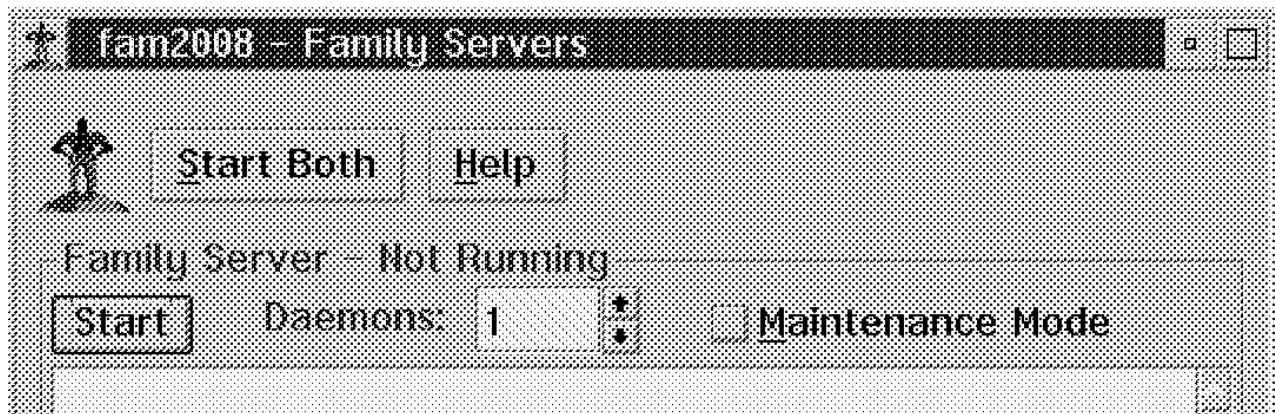


Figure 33. Family Servers Window

6.0 Chapter 6. Using TeamConnection

```
"This is it boys - over the hill."
-- John Lennon --
```

Using TeamConnection is an evolutionary process. If you are a small development project, a first-time user, or in the early stage of a development project, you will probably use only the very basic features of TeamConnection, such as versioning. If, however, you are an experienced user, a large development group, in the final stage of a development project, or maintaining an existing application or system, you probably want to take advantage of the more rigorous process management that TeamConnection offers.

Whoever you are, all users of TeamConnection have to go through some basic steps before they can start using TeamConnection. In this chapter we take you through these basics:

- ☐ Creating user IDs and host lists
- ☐ Creating components
- ☐ Creating a release
- ☐ Creating a work area
- ☐ Creating parts
- ☐ Working with parts
- ☐ Creating versions

In Chapter 9, "Using TeamConnection's Integrated Problem Tracking and Change Control" in topic 9.0 we take a closer look at using the different TeamConnection process management options and such functions as defects, features, tracking, and drivers. But first things first. We begin by explaining how to start the TeamConnection client and take a look at some of the basic settings for the client GUI.

```
+--- Notational convention -----+
|
| When we reference items in the GUI, we reference them in the following
| way:
|
|
| Bold face text
|     Menu items are presented in boldface, for example, Selected.
|
|
| Windows >
|     If a menu item is followed by an arrow (>), additional
|     submenu items are available.
|
|
| TeamConnection - Tasks
|     The various windows themselves are referenced with the
|     "example" font: Help.
|
+-----+
```

Subtopics

- 6.1 Starting the TeamConnection Client
- 6.2 Creating TeamConnection Users and Host Lists
- 6.3 Creating Components
- 6.4 Creating a Release
- 6.5 Creating a Work Area
- 6.6 Creating Parts
- 6.7 Working with Parts
- 6.8 Creating Versions

6.1 Starting the TeamConnection Client

Before you start the TeamConnection client, you should have read and carried out the steps outlined in Chapter 4, "Installing TeamConnection" in topic 4.0 . If you have installed TeamConnection successfully, you should have a folder on your desktop that looks like this: PICTURE 36

Double-click on this folder to open up the TeamConnection Group folder. In the TeamConnection Group folder, double-click on the TeamConnection Client icon PICTURE 37 to start the client.

Subtopics

6.1.1 TeamConnection - Tasks Window

6.1.2 The Settings Notebook

6.1.1 TeamConnection - Tasks Window

The first window you see when starting the TeamConnection client is the **TeamConnection - Tasks** window (see Figure 34). This window is like your "desktop" for TeamConnection, and you use it as the starting point for all of your work in TeamConnection when using the GUI. The **TeamConnection - Tasks** window comes with a basic setup, but it can be configured to your needs. Information about the client setting is stored in a file called **TEAMC10.INI**. This file is in the *drive:\OS2* directory, but you can find a copy of it in the *drive:\TEAMC\BIN* directory. We highly recommend that you save your *customized* version of the **TEAMC10.INI** file in a separate directory, so that you can restore it if for some reason the existing copy should be corrupted (it is always very frustrating to re-create setups).

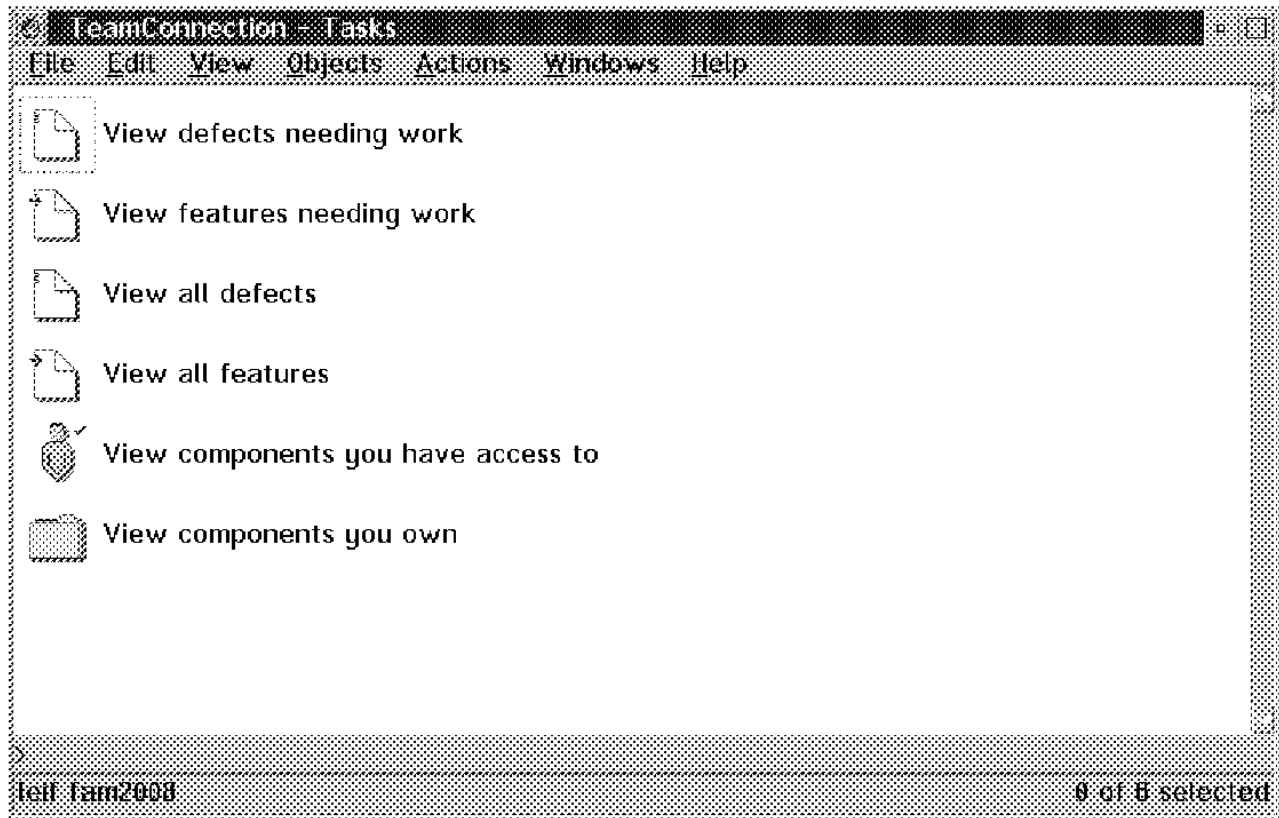


Figure 34. TeamConnection - Tasks Window

The **TeamConnection - Tasks** window has several menu items:

File

The **File** menu item has the following submenu items:

Task list

This menu item enables you to modify your **TeamConnection - Tasks** window. For example, you can move and change the position of the tasks already defined in the window, you can insert blank lines, and you can edit the *Tasks list*, which means that you can add, delete, or modify tasks in the **Tasks** window.

Print

Prints the current task list

Minimize all

Minimizes all open windows and presents them as icons

Close other Windows

Closes all open windows except for the current window

Exit

Select this item to exit the TeamConnection client GUI.

Edit

The **Edit** menu item has the following submenu items:

Execute

Runs the selected task. You can also run the task by double-clicking on the task description or task icon.

Modify

Displays the **Edit Task List** window so that you can modify the selected task

Delete

Deletes the task you have selected

Select all

Selects all items in the list

Deselect all

Deselects items you have selected

View

The **View** menu item has the following submenu items:

Text

Displays list item descriptions only

Icon

Displays both descriptions and their corresponding icons

Flowed icon

Displays the list of icons arranged in columns across the screen

Details

Displays detailed information about list items

Save view as default

Saves the view of the current window so that it is always displayed in the same way. For example, if you select this choice when the Icon view is selected, the window always displays with objects represented as icons.

Save window size/position

Saves the current size and position of the window from one session to the next

Reorder columns

Enables you to specify the order in which the window columns are displayed in the Details view

Objects

The **Objects** menu item has the following submenu items:

Builders

Displays the **Builders** window that enables you to define the specific tools that are needed for your development environment

Change history

Displays the change history information for releases, drivers, and parts

Components >

Enables you to work with components, access lists, or notification lists:

Components

Displays the **Components** window that enables you to group parts in a release, control user access to parts, and control user notification of TeamConnection events

Access lists

Displays the **Access Lists** window that enables you to associate users with authority groups

Notification lists

Displays the **Notification Lists** window that enables you to associate users with interest groups

Defects >

Enables you to work with defects, sizing records for defects, or defect verification records:

Defects

Displays the **Defects** window that enables you to work with defects

Sizing records

Displays the **Sizing Records** window that enables you to indicate the time and resources needed to resolve one defect or implement one feature in one component for one release

Verification records

Displays the **Verification Records** window that enables you to work with verification records

Drivers >

Enables you to work with drivers, driver members, or the driver change history:

Drivers

Displays the **Drivers** window that enables you to work with drivers

Driver members

Displays the **Driver Members** window that enables you to work with the members in a specific driver

Change history

Displays the **Driver Change History** window that enables you to display a graphical view of the objects in which drivers have changed over time

Features >

Enables you to work with features, sizing records for features, or feature verification records:

Features

Displays the **Features** window that enables you to work with features

Sizing records

Displays the **Sizing Records** window that enables you to indicate the time and resources needed to resolve one defect or implement one feature in one component for one release

Verification records

Displays the **Verification Records** window that enables you to work with verification records.

Parsers

Displays the **Parsers** window that enables you to extract dependency information from build objects.

Parts >

Enables you to work with parts, view a build tree structure, perform work area actions, perform build operations, work with collision records, or view the part change history:

Parts

Displays the **Parts** window that enables you to work with parts

BuildView

Displays the **Build View** window that enables you to see parts in a build tree structure

PartFull

Displays the **PartFull** window that enables you to work with parts, perform work area

actions, or perform a build operation

Collision records

Displays the **Collision Records** window that enables you to work with collision records

Change history

Displays the **Part Change History** window that enables you to display a graphical view of the objects in which parts have changed over time

Releases >

Enables you to work with releases, approver lists, and environment lists:

Releases

Displays the **Releases** window that enables you to work with parts that are part of a specific release

Approver lists

Displays the **Approver Lists** window that enables you to work with the approver list for a release

Environment lists

Displays the **Environment Lists** window that enables you to work with the environment list for a release

Users >

Enables you to work with user IDs and host lists:

Users

Displays the **Users** window that enables you to work with user IDs

Host lists

Displays the **Host Lists** window that enables you to work with host lists

Versions

Displays the **Version** window that enables you to view current and previous versions of TeamConnection parts

Work areas >

Enables you to work with work areas, approval records, fix records and test records:

Work areas

Displays the **Work Area** window that enables you to work with work areas

Approval records

Displays the **Approval Records** window that enables you to work with approval records for a work area

Fix records

Displays the **Fix Records** window that enables you to work with fix records for a work area

Test records

Displays the **Test Records** window that enables you to work with test records for a work area

Command window

Displays the **Command Window** that enables you to enter commands and view the results.

Actions

The **Actions** menu item has the following submenu items:

Builders >

Select Builders to work with builders and build objects and associate builders with build properties.

Components >

Select Components to work with components, view information about components, and work with component defects and features.

Defect >

Select Defect to work with defects, view defect information, change the state of defects, and work with the process configuration, sizing, and design information regarding defects.

Features >

Select Feature to work with features, view feature information, change the state of features, and work with the process configuration, sizing, and design information regarding features.

Drivers >

Select Drivers to work with drivers.

Parsers >

Select Parsers to work with parsers.

Parts >

Select Parts to work with parts, build information, work areas, and work with builds and build objects.

Releases >

Select Releases to work with releases, release information, and release process configurations, add approvers to the approver list, and create work areas.

Users >

Select Users to work with users, user IDs, user properties, and host lists.

Versions >

Select Versions to work with versions.

Work areas >

Select Work areas to work with work areas, corequisites, and driver members and create approval and fix records.

Lists >

Select Lists to work with various lists.

Records >

Select Records to work with various records.

Windows

The names of all open windows appear as selectable choices in this menu. Consequently, the menu choices change dynamically as you open and close windows. The **Settings** choice, however, always appears in this menu. Select this choice to open the **Settings** notebook.

Help

Gives access to the **Help** submenu items.

Logging the GUI commands

When you are working with the GUI, TeamConnection translates and logs every command that is issued from the GUI. The commands are translated into command line interface syntax and stored in a log file. This file is specified in the **Log file** field in the **Setup** page in the **Settings** notebook (see "The Settings Notebook" in topic 6.1.2). The default name is *teamc.log* and is stored in the *\BIN* directory where the client is installed.

6.1.2 The Settings Notebook

As a first-time user of TeamConnection, the first thing that you should do is update the settings pages in the **Settings** notebook. You access the **Settings** notebook by selecting menu items **Windows > Settings**. You should then get a window like that shown in Figure 35.

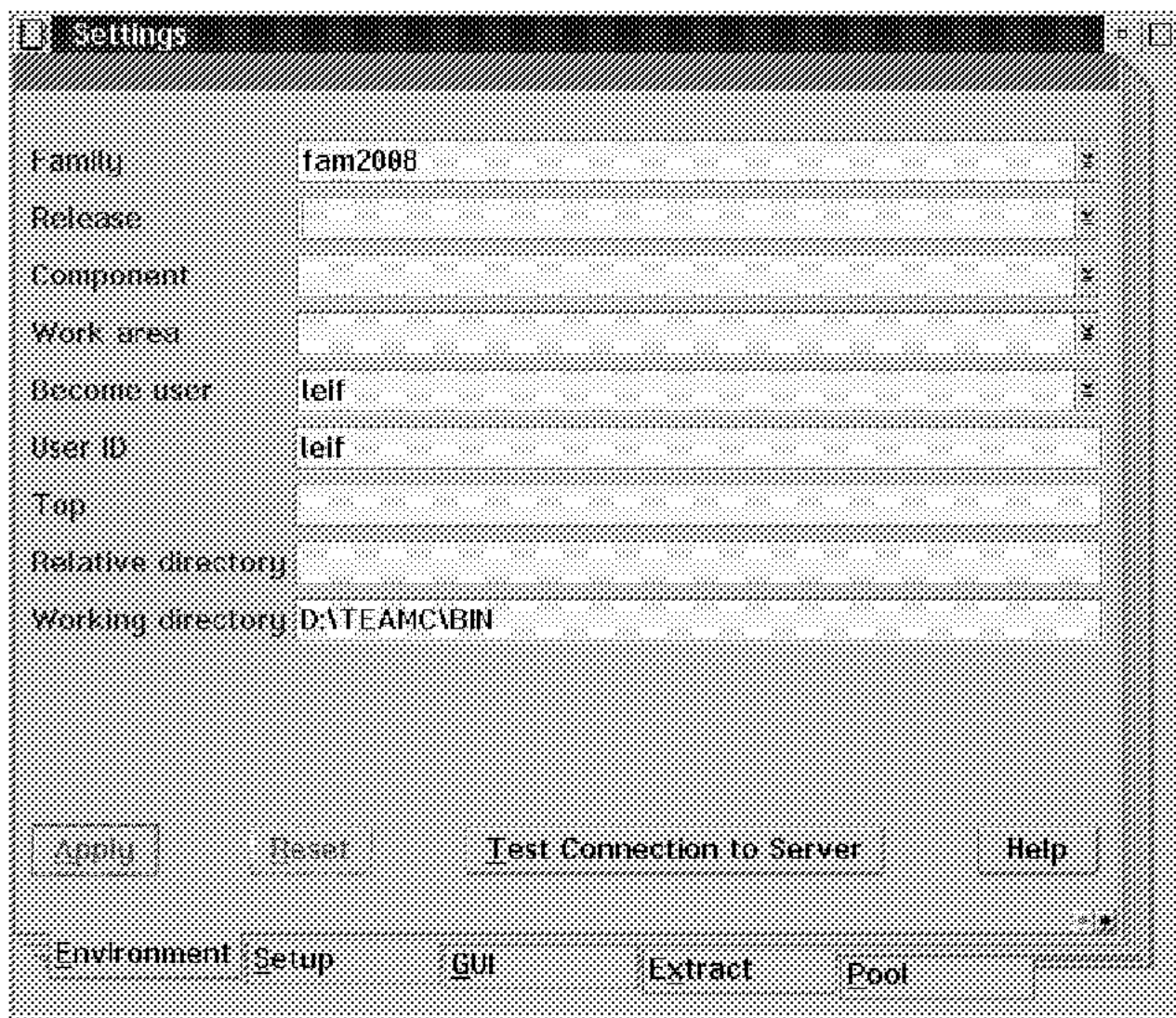


Figure 35. Settings Window

As you can see in Figure 35, the **Settings** notebook consists of five pages, namely:

- ☐ Environment
- ☐ Setup
- ☐ GUI
- ☐ Extract
- ☐ Pool

Subtopics

- 6.1.2.1 Environment Page
- 6.1.2.2 Setup Page
- 6.1.2.3 GUI Page
- 6.1.2.4 Extract Page
- 6.1.2.5 Pool Page

6.1.2.1 Environment Page

Use the **Environment** page (see Figure 35 in topic 6.1.2) to set the default values for the following parameters:

Family

Type the name of the TeamConnection family with which you want to work. This sets the TC_FAMILY environment variable. The syntax for this field is *family@server@port*. Contact your family administrator if you do not know the information to type in this field.

Release

Type the release that you will work with most often. This sets the TC_RELEASE environment variable, which applies to commands that you enter on the command line of the **Tasks** window or **TeamConnection - Commands** window. In addition, this release name appears in the Release field when certain windows are displayed.

Component

Type the component that you will work with most often. This sets the TC_COMPONENT environment variable which applies to commands that you enter on the command line of the **Tasks** window or **TeamConnection - Commands** window. In addition, this component name appears in the Component field when certain windows are displayed.

Work area

Type the work area that you will use most often. This sets the TC_WORKAREA environment variable, which applies to commands that you enter on the command line of the **Tasks** window or **TeamConnection - Commands** window.

Become user

Type the user ID from which you want to issue commands if the user ID differs from the user ID you logged on with. This sets the TC_BECOME environment variable. You own the access authority of the user ID you specify. If you do not specify this variable, TeamConnection uses the value you specify in the User ID field.

User ID

Type the TeamConnection user ID that your family administrator assigned to you. This sets the TC_USER environment variable.

Top

Type the leading portion of the path name that is a subset of the current working directory on your client machine. This sets the TC_TOP environment variable. TeamConnection does not use the value in this field when you specify a value in the Relative directory field.

Relative directory

Type the name of the drive and directory that you will use most often when you create, copy, and update parts. This directory name appears in the Directory field when certain windows are displayed. If you leave this field empty, these Directory fields are left blank and TeamConnection uses the Working directory field.

Working directory

The default directory that TeamConnection uses to complete your requests. This applies to commands that you enter on the command line of the **Tasks** window or **TeamConnection - Commands** window.

From the **Environment** page, you can also test the connection to the server that you want to access, by filling in the name of the family and clicking on the **Test Connection to Server** button (see Figure 35 in topic 6.1.2). If the connection is in order, this will be displayed in an information window like that shown in Figure 36.

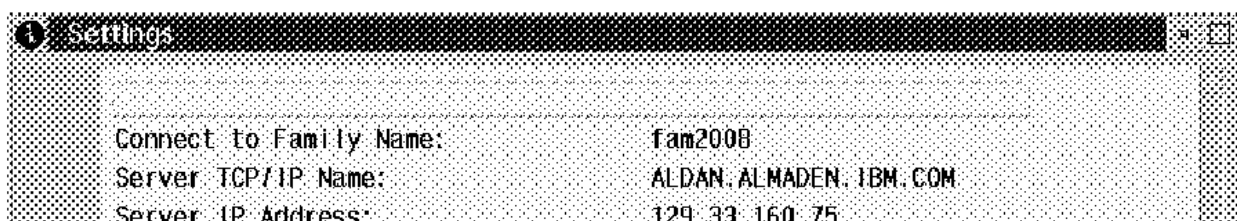


Figure 36. Information Window for Testing Connection to Server

6.1.2.2 Setup Page

Use the **Setup** page (see Figure 37) to set the default values for the following parameters:

NLS path

Type the path that points to your *TC.CAT* file followed by a required parameter (%N). You must type the %N in uppercase. For example, if *TC.CAT* is located in *C:\LIB\EXE*, type:

`C:\LIB\EXE\%N`

in this field. This sets the `TC_NLSPATH` environment variable.

Log file

Type the name of the file where you want the TeamConnection GUI to store the commands that it issues. The default is *teamc.log*. Use your favorite editor to prune this file when it becomes too large. If you do not specify a log file name, TeamConnection does not store the commands you issue.

Case

Select a radio button to specify that the arguments in the TeamConnection commands should be as follows:

- ☐ Left as you typed them (Ignore)
- ☐ Converted to uppercase (Upper)
- ☐ Converted to lowercase (Lower)

This sets the `TC_CASESENSE` environment variable, which applies only to arguments in commands, not in queries. You generate queries when you use a filter window and commands when you use other TeamConnection windows. For example, if you use the **Parts Filter** window to find a part called *myPart.doc*, you must type the part name in the same case that was used when the part was first created in the TeamConnection database.

Print command

Type the command you want to use to invoke your print program from within TeamConnection.

Compare command

Type the command you want to use to invoke your compare program from within TeamConnection. For example, if the compare program you want to use is invoked by typing **comp**, type **comp** in this field. The data you type also appears in the **Show Part Differences** window.

Edit command

Type the command you want to use to invoke your editor from within TeamConnection. For example, if you invoke your editor by typing **E**, type **E** in this field. The command you type also appears in the **Edit Part** window.

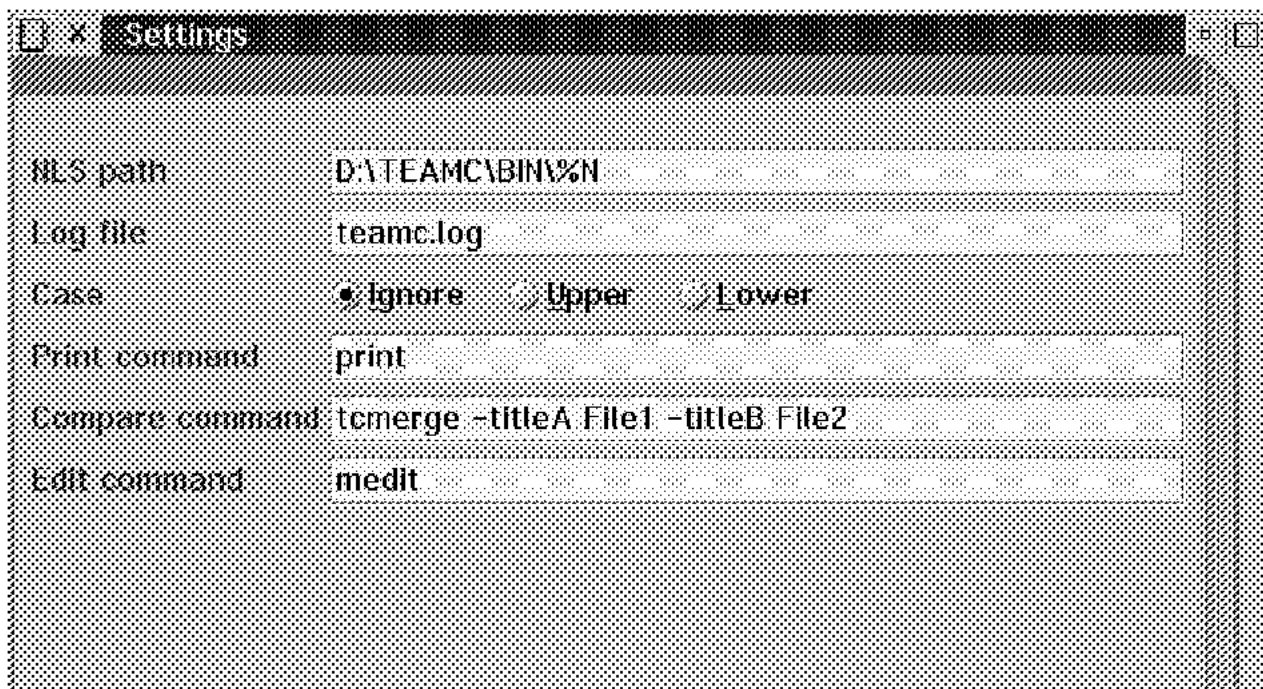


Figure 37. Setup Page

6.1.2.3 GUI Page

Use the **GUI** page (see Figure 38) to set the default view for every TeamConnection window that you open.

Verbose commands

Displays a detailed account of TeamConnection actions. For example, if the verbose action is selected and you successfully extract a part, you get a message stating that the part was successfully extracted. You get no such message if the verbose action is not selected. This does not affect error messages.

Auto refresh

Provides an automatic refresh of the current window after various actions occur. If you do not select this check box, you can manually refresh a list by either selecting Refresh now from the Part pull-down menu or pressing F5.

Note: If an error occurs during the auto refresh, the refresh command stops, and the refresh does not occur.

Multiple object windows

Enables you to display more than one object window at the same time. To see which windows you have open, select the **Windows** menu.

Show query line

Displays the current query line in object windows. When the query line is displayed, you can change the query and then press Enter to start it. To undo your changes, press Esc.

Sort pre-defined list values

Alphabetically sorts the predefined values in the following fields:

- ☐ Authority on Access dialogs
- ☐ Interest on Notification dialogs
- ☐ Process on Component and Release dialogs
- ☐ Type on Driver dialogs
- ☐ Configurable fields on Defect, Feature, Part and User dialogs

Use small icons in icon views

Uses small icons in the icon view. This is useful when you want to see more icons at one time.

Use small icons in tree views

Uses small icons in the tree view. This is useful when you want to see more icons at one time.

Font for object windows

Enables you to change the font, style, and size of text that appears in object windows. Select the **Choices** push button for a list of available fonts.

Font for output windows

Enables you to change the font, style, and size of text that appears in the **Information** or the **Command** window. Select the **Choices** push button for a list of available fonts.

Required field label color

Enables you to customize the color of field labels for fields that require input. The default is dark pink.

Modified field label color

Enables you to customize the color of field labels for fields that you modify. The color of the label changes after you type data in the field. The default color is green.

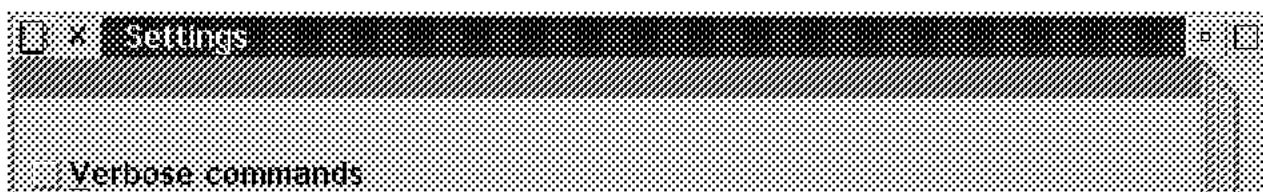


Figure 38. GUI Page

6.1.2.4 Extract Page

Use the **Extract** page (see Figure 39) to set the default view for windows used to extract releases or extract drivers.

Destination directory

Name of the directory on the TeamConnection client where TeamConnection will place the extracted part trees. Part trees are directory structures representing the parts in a release.

Read-only

Select this check box to specify that the extracted parts are read-only. You can override the default by using octal notation to specify the file permissions you want.

Expand keywords

Select this check box to substitute assigned values in place of keywords. The default is to substitute values for keywords. This is valid only for parts stored as text in TeamConnection. If the part is stored as binary, TeamConnection does not expand the keywords. Selecting this check box also changes the default value of the Expand keywords check box in windows that perform extracts. These windows are the **Extract Files**, **View File Contents**, and **Show File Differences** windows. For more information, refer to "Supported Keywords" in the *IBM TeamConnection for OS/2 User's Guide*, SC34-4499.

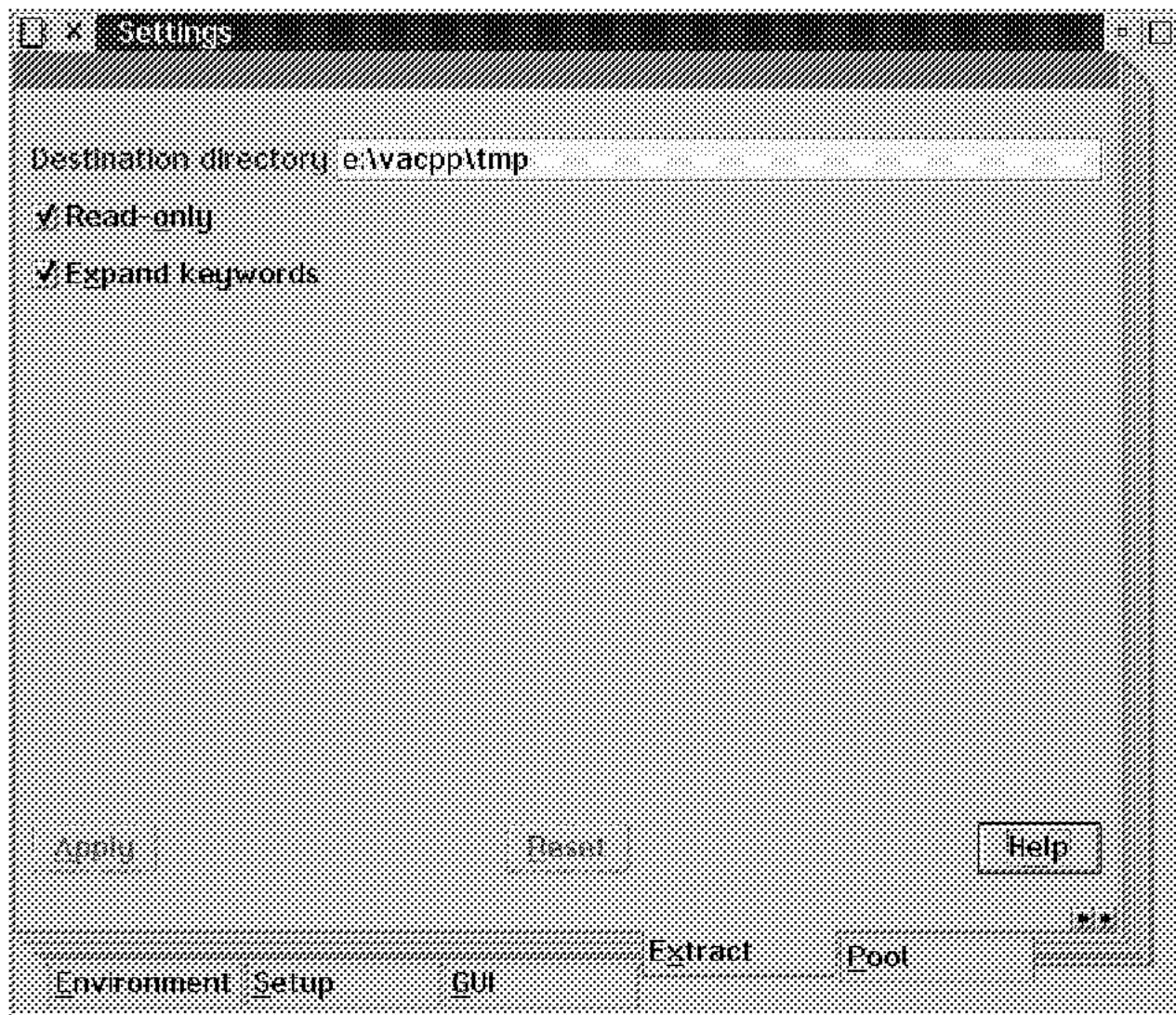


Figure 39. Extract Page

6.1.2.5 Pool Page

On this page (see Figure 40) you specify the name of the build pool that you want to be the default when you use the build function. The build pool defines the set of build agents that should be used to process the build request.

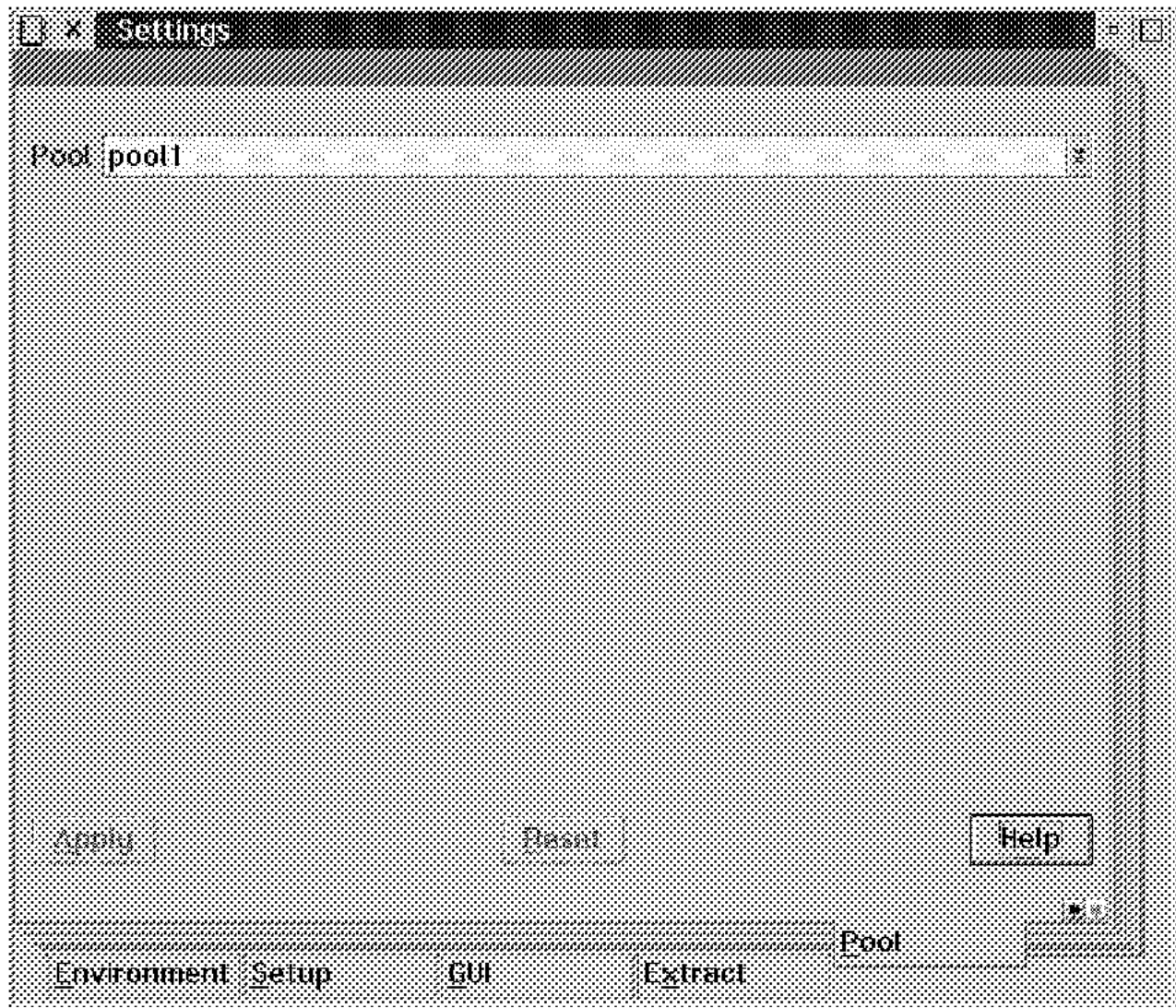


Figure 40. Pool Page

6.2 Creating TeamConnection Users and Host Lists

Creating a user in TeamConnection can be done by using either the GUI or the command line interface. If you have to create many users, the command line interface is preferable. The same is true for any repetitive task that you have to perform in the family.

In the sections that follow we show you how to create users and host lists by using both the GUI and the command line interface. Remember that you have to be a *superuser* to create users and host lists.

Subtopics

6.2.1 Using the GUI

6.2.2 Using the Command Line Interface

6.2.1 Using the GUI

There are several ways in which you can create user and host lists entries in TeamConnection using the GUI:

- ☐ From the **TeamConnection - Tasks** window by selecting **Actions > Users > Create...**
- ☐ From the **TeamConnection - Users** window by selecting **Selected > Create...**

You can get to the **TeamConnection - Users** window by selecting **Objects > Users > Users...** from the **TeamConnection - Tasks** window or from a task in your **TeamConnection - Tasks** window (if you created a task).

If you select **Objects > Users > Users...** from the **TeamConnection - Tasks** window, you will get a window that looks like that in Figure 41. Use the **User Filter** window to search for specific user lists. The data you type in the filter fields is case-sensitive; you must type it exactly as it was entered in the database. You can type more than one item in some of these fields. If you do so, separate the items with a space. The following is an explanation of the different fields in the **User Filter** window:

User IDs

Type the TeamConnection user IDs that you want to list.

User names

Type the names of the users.

User areas

Type the departments or areas in which the users work. If you want to remove the user areas from a parser, type NULL in this field.

User mail addresses

Type the mailing address where the user's TeamConnection mail is to be sent. The mail address must not exceed 143 single-byte characters. Use the following format:

login@hostname

Add dates

Type the dates (yyyy/mm/dd) on which user IDs were created.

Delete dates

Type the dates (yyyy/mm/dd) on which user IDs were deleted.

Change dates

Type the dates (yyyy/mm/dd) on which user IDs or information associated with the user IDs were last changed.

Superuser

Select one of the following radio buttons:

Yes

Lists users that have superuser privilege

No

Lists users that do not have superuser privilege

Both

Lists users regardless of whether they have superuser privilege

History

A list of previous queries created by using this **User Filter** window

Query

The query you are about to submit

Note: Your family administrator can configure additional fields for the window. TeamConnection help is not available for any additional fields that your family administrator creates.



Figure 41. User Filter Window

After having completed the selection by clicking on the **OK** push button, you get a window that looks like that in Figure 42, which shows the *Details* view format. The default format for this window is the *Icon* view format. You can change the view format by selecting the **View** menu item and from there selecting any of the following view formats:

Text

Displays list item descriptions only

Icon

Displays both descriptions and their corresponding icons

Flowed icon

Displays the list of icons arranged in columns across the screen

Details

Displays detailed information about list items

The following are other **View** menu items that also affect the behavior of the window:

Save view as default

Saves the view of the current window so that it is always displayed in the same way. For example, if you select this choice when the **Icon** view is selected, the window always displays with objects represented as icons.

Save window size/position

Saves the current size and position of the window from one session to the next

Reorder columns

Enables you to specify the order in which the window columns are displayed in the **Details** view

TeamConnection - Users			
File Selected Edit View Objects Windows Help			
User ID	User Name	Area	Mail Address
InheritedAccess			
leif	Leif Trulsson	BLD80/E2_RM412	leif@aldan.almaden.ibm.com
vgts	Leif Trulsson	BLD80/E2_RM412	vgts@aldan.almaden.ibm.com
stade5	Lutz Sparmann	BLD80/E2_RM255	stade5@sthelens.almaden.ibm.com
paul	Paul Fowler	BLD70B_RM51B	paul@ontario.almaden.ibm.com
ulf	Ulf Flodén	BLD70B_RM59B	stade5@indus.almaden.ibm.com
mehdi	Mehdi Sanayi	BLD70B_RM45B	mehdi@cubango.almaden.ibm.com
stade6	Yuhuske Watanabe	BLD80/E2_RM255	stade6@sumatra.almaden.ibm.com

1=1
leif fam2008 0 of 8 selected

Figure 42. TeamConnection - Users Window

By selecting **Selected > Create...** from the **TeamConnection - Users** window or **Actions > Users > Create...** from the **TeamConnection - Tasks** window, you open up the **Create User** window (see Figure 43).

Create User	
User ID	felix
Mail address	felix@aldan.almaden.ibm.com
Full name	Felix Trulsson
Area	BLD80/E2_RM255
<input type="checkbox"/> Superuser	
<input type="button" value="Ok"/> <input type="button" value="Apply"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>	

Figure 43. Create User Window

Use the **Create User** window to create new users in the database. A new user has authority to open defects or features against any component in the family. When creating a new user on the database, you must supply a user ID and a mail address. You can also specify the user's full name and area. Only a superuser can create a new user for a family. The following is an explanation of the fields used:

User ID

Type the user ID of the users you are adding. You can use any alphanumeric characters (a-z, A-Z, 0-9), plus ordinary punctuation. Do not use the dollar sign (\$), double or single quotes, or the \ character.

Mail address

Type the mailing address where the user's TeamConnection mail is to be sent. The mail address must not exceed 143 single-byte characters. Use the following format:

login@hostname

If you type just the host name, the login is assumed to be the same as the user ID.

Full name

Type the user's last name, first name, and initial. You can type up to 31 characters, using any alphanumeric characters (a-z, A-Z, 0-9), plus ordinary punctuation. Do not use the dollar sign (\$), double or single quotes, or the back slash (\) character

Area

Type any text string here to identify the part of your organization to which the user belongs, such as department or project code. You can use any alphanumeric characters (a-z, A-Z, 0-9), plus ordinary punctuation. Do not use the dollar sign (\$), double or single quotes, or the \ character.

Superuser

Select this if the new user is to have superuser authority. A superuser has authority to perform all TeamConnection actions. Your current TeamConnection user ID must have superuser privileges in order to grant another user these privileges.

Note: Your family administrator can configure additional fields for the window. TeamConnection help is not available for any additional fields that your family administrator creates.

Use the **Apply** push button to process the information you typed and leave the window open (consecutive creates) and use the **OK** push button to process the information that you typed and close the window.

Each TeamConnection user ID is associated with a host list, which is a list of host names from which the user can access TeamConnection. The host name can use either of the following formats:

- ☐ host name
- ☐ host name.address (where address is the TCP/IP address)

The TeamConnection server uses the host list to verify the user's authenticity. Use the access list to specify a user's authorization. There must be at least one host list entry for each TeamConnection user ID. Each entry consists of a login, a TeamConnection user ID, and a host name. For example, a user has the following entry in his host list:

Login	Host Name	User ID
ProjLead	dev1.edu.com	dev1

The user's login is *ProjLead*, which associates his TeamConnection user ID, *dev1*, with his host name, *dev1.edu.com*.

You can create a host list entry in several ways:

- ☐ By selecting an entry in the **TeamConnection - Users** window, clicking on that entry with the right mouse button (see Figure 44) to get the pop-up menu, and selecting the **Add Host** menu item
- ☐ By selecting the **Selected > Add Host...** from the **TeamConnection - Users** window
- ☐ By selecting **Selected > Add...** from the **TeamConnection - Hosts Lists** window
- ☐ By selecting **Actions > Users > Add host...** from the **TeamConnection - Tasks** window

User ID	User Name	Area	Mail Address
InheritedAccess leif	Leif Trulsson	BLD80/E2_RM412	leif@aldan.almaden.ibm.com

Figure 44. TeamConnection - Users Window with Pop-Up Menu

Whichever way you choose, you will get the **Add Host** window as shown in Figure 45. Use the **Add Host** window to add a host name to a host list. Each host name identifies a TeamConnection client from which each user ID is authorized to access the family server. A user can have multiple host entries. Only the owner of the user ID or a superuser can add a host name. Someone with TeamConnection superuser privilege must create the initial host list entry for each user.

The **Add Host** window has the following fields:

Login

Type the ID for the user that is to have access to TeamConnection. This is your system login. The login is case sensitive.

Host name

Type the name of the client machine from which the user ID can access the family server. This is your TeamConnection ID. You can use either of the following formats:

- ☐ host name
- ☐ host name.address (where *address* is the TCP/IP address)

User ID

Type the TeamConnection user ID for the user who is to have access from the specified host. You can import a selected user from the Users window.

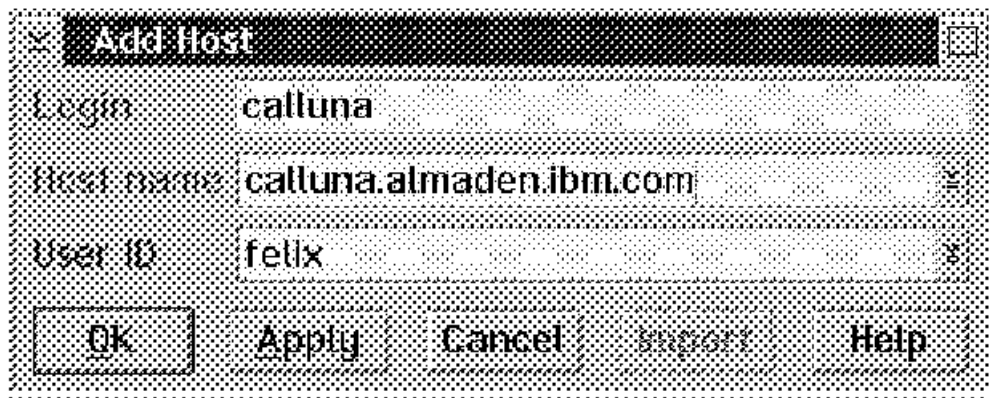


Figure 45. Add Host Window

To access the **TeamConnection - Host Lists** window (see Figure 46), select **Objects > Users > Host lists...** from the **TeamConnection - Tasks** window, or if you have create an entry for your host lists in your task list, select that entry. From the **TeamConnection - Host Lists** window you can add, delete, or modify host list entries.

TeamConnection - Host Lists				
File	Selected	Edit	View	Objects
Login	Host Name	User ID	User Name	Area
mehdi	cubango.almaden.ibm.com	mehdi	Mehdi Sanayei	BLD70B_RM45B
ulf	indus.almaden.ibm.com	ulf	Ulf Flodén	BLD70B_RM59B
paul	ontario.almaden.ibm.com	paul	Paul Fowler	BLD70B_RM51B
sumatra	sumatra.almaden.ibm.com	stade6	Yuhstake Watanabe	BLD80/E2_RM255
stade5	sthelens.almaden.ibm.com	stade5	Lutz Sparmann	BLD80/E2_RM255
stade5	sthelens.almaden.ibm.com	leif	Leif Trulsson	BLD80/E2_RM412
leif	aldan.almaden.ibm.com	leif	Leif Trulsson	BLD80/E2_RM412
leif	itscsrv2.almaden.ibm.com	leif	Leif Trulsson	BLD80/E2_RM412
leif	itscsrv2.almaden.ibm.com	vgts	Leif Trulsson	BLD80/E2_RM412
leif	calluna.almaden.ibm.com	leif	Leif Trulsson	BLD80/E2_RM412
leif	itscsrv2.almaden.ibm.com	leif	Leif Trulsson	BLD80/E2_RM412

Figure 46. TeamConnection - Host Lists Window

6.2.2 Using the Command Line Interface

Use the **user** command to create new user IDs, modify information associated with user IDs, and delete user IDs. Superuser privilege is required to create user IDs for new users, delete other user IDs, and modify the superuser privilege of a user. You can modify your own user ID information but cannot give yourself TeamConnection superuser privilege.

If you want to use the TeamConnection command line interface commands to create your users and host lists, simply create a .CMD file similar to that shown in Figure 47 and then run the file.

```
+-----+
|
| teamc User -create -login parkin -address calluna -name "Kay Parkin"
| teamc User -create -login goti -address calluna -name "Carlos Goti"
| teamc User -create -login mccart -address calluna -name "Pat McCart"
| teamc User -create -login koch -address calluna -name "Henry Koch"
| teamc User -create -login eber -address calluna -name "Jim Eberwein"
| teamc User -create -login gauck -address calluna -name "Dave Gauck"
| teamc User -create -login orlo -address calluna -name "Tim Orlowski"
|
+-----+
```

Figure 47. Creating Users through the Command Line Interface

Although the **user** command establishes required user ID information, actual host access for a user ID must be created by using the **host** command. A similar .CMD file for the creation of host lists would look like that in Figure 48. In this example we create a host list for host calluna. As you can see, we allow more than one user to access TeamConnection from host calluna.

```
+-----+
|
| teamc Host -create parkin@calluna -login parkin
| teamc Host -create goti@calluna -login goti
| teamc Host -create mccart@calluna -login mccart
| teamc Host -create koch@calluna -login koch
| teamc Host -create eber@calluna -login eber
| teamc Host -create gauck@calluna -login gauck
| teamc Host -create orlo@calluna -login orlo
|
+-----+
```

Figure 48. Creating a Host List through the Command Line Interface

You can of course combine both files into one single file.

6.3 Creating Components

A *component* is a TeamConnection object that simplifies project management, organizes project data into structured groups, and controls configuration management properties.

Within each family, development data is organized into groups of *components*. The component hierarchy of each family includes a single top component, initially called *root*, and descendants of that root. Each child component has at least one parent component; a child can have multiple parents.

TeamConnection uses components to organize development data, control access to the data, and notify users when certain actions occur. Descendant components inherit access and notification information from ancestor components. Information about the components is stored in the database, including:

- Component's position in its family hierarchy.
- User who owns the component. The component owner is responsible for managing data related to it, including problems or design changes.
- Users who have access to the component and the level of user access. This information makes up the component's *access list*.
- Users who are to be notified about changes to the component. This set of users is called the *notification list*.

In the sections that follow we show you how to create components by using both the GUI and the command line interface.

Who can create components?

To initially create components in TeamConnection, you have to be a **superuser**. Typically, the family administrator creates the initial components and then transfers ownership of the components to, for example, a project leader or design leader. Once the component ownership has been transferred, the new owner can then create new child components and transfer ownership of the child components to other members of the organization or development team if necessary.

The new owner can also create **access lists** and **notification lists**. Therefore, for example, the owner of the **data** component (typically the database administrator in a larger organization) could grant access and notification to this component and child components, if any, so that only authorized persons can change the data definitions, for example.

Subtopics

6.3.1 Using the GUI

6.3.2 Transfer Component Ownership

6.3.3 Using the Command Line Interface

6.3.1 Using the GUI

As with the creation of user and host lists (see "Using the GUI" in topic 6.2.1), there are several ways in which you can create components. You can choose from the following methods to get to the **Create Components** window:

- ☐ By selecting **Actions > Components > Create...** from the **TeamConnection - Tasks** window
- ☐ By selecting **Selected > Create...** or **Selected > Create child...** from the **TeamConnection - Components** window
- ☐ By selecting a component in the **TeamConnection - Components** window, clicking with the right mouse button on the selected component to get the pop-up menu, and then selecting **Create child...** from there.

You can get to the **TeamConnection - Components** window by either **Objects > Components > Components...** from either the **TeamConnection - Tasks** window or a task in your **TeamConnection - Tasks** window (if you created a task). If you select **Objects > Components > Components...** from the **TeamConnection - Tasks** window, you will first get the **Component Filter** window (see Figure 49).

Use the **Component Filter** window to search for components. The data you type in the filter fields is case sensitive; you must type it exactly as it was entered in the database. You can type more than one item in some of these fields. If you do so, separate the items with a space. Here is an explanation of the fields in the **Component Filter** window:

Components

The names of the components that you want listed

Owner IDs

The user IDs of the owners of the components

Owner names

The names of the component owners

Owner areas

The departments or areas in which the component owners work

Processes

The names of the processes that were defined for each component. You can use the down arrow to view and select from a list of available choices.

Add dates

Dates (yyyy/mm/dd) on which the components were created

Delete dates

Dates (yyyy/mm/dd) on which the components were deleted

Change dates

Dates (yyyy/mm/dd) on which the components or information associated with the components were last changed

Descriptions

Information describing the purpose of the components

Use the radio buttons to indicate whether you want to list components that have the characteristic or subprocess listed. For example, let's look at the DSR subprocess. If you select:

Yes

Lists only components with the feature DSR subprocess

No

Lists only components without the feature DSR subprocess

Both

Lists components regardless of whether the feature DSR subprocess is defined

Use the previous example for each of the following:

Feature DSR subprocess

Components that have a DSR subprocess for all features

Feature Verify subprocess

Components that have a verify subprocess for all features

Defect DSR subprocess

Components that have a DSR subprocess for all defects

History

A list of previous queries created by using this **Component Filter** window

Query

The query you are about to submit

Figure 49. Component Filter Window

Note: Figure 49 displays two windows to show all fields available in the **Components Filter** window.

Use the **Create Components** window (see Figure 50) to create new components. If you create a child component, the parent component appears in the Parent field.

If you create a component, you become its owner and have authority to create an access list and a notification list for that component.

If you receive a message saying that you do not have the necessary authority, contact your family administrator and request *CompCreate* authority.

You can type more than one item in some of these fields. If you do so, separate the items with a space. Many fields provide a pull-down menu from which you can select items for the field. The pull-down contains a list of up to the last 10 values you previously entered in this field. If the list does not contain the value you want to use, select the **Show all choices** push button at the bottom of the list to get a list of all of the valid choices for this field. To access the pull-down menu, either select the down arrow at the right of the field or press F7.

The following is a description of the fields in the **Create Components** window:

Components

Type the names of the components you want to create. A component name cannot exceed 31 characters. You cannot use the name of an existing component in your family or the name of a component that has been removed from active status. You can use any alphanumeric characters (a-z, A-Z, 0-9) and ordinary punctuation. Do not use the dollar sign (\$), double or single quotes, or the \ character.

Parent

Type the name of the component that is the parent of the component you are creating. Although a component can have more

than one parent, you can specify only one here. You can use the link components action to specify additional parents.

Process

Specify a *process* for the component. The following processes are supplied with TeamConnection:

- ☐ default
- ☐ prototype
- ☐ development
- ☐ test
- ☐ preship
- ☐ maintenance
- ☐ emergency_fix

Your family administrator might have removed, added to, or changed these processes.

Owner

You can type the user ID of the person to whom you want to assign ownership. If you do not specify a name, you will be the component owner. The owner has authority to define an access list and a notification list for that component. You can use the **Import** push button to retrieve the user ID of a selected user from the **TeamConnection - Users** window.

Description

You can type text to describe the purpose of the component. The maximum length is 63 characters. You can use any alphanumeric characters (a-z, A-Z, 0-9) and ordinary punctuation. Do not use the dollar sign (\$), double or single quotes, or the \ character.

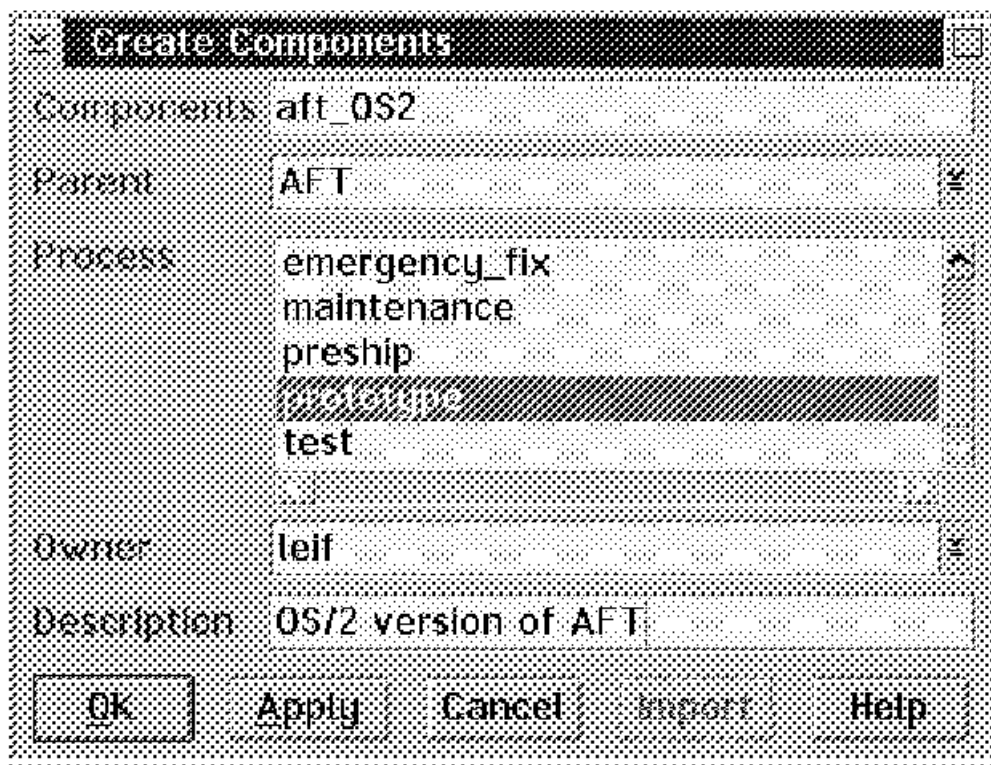


Figure 50. Create Components Window

As you can see in Figure 50, we chose the *prototype* process for this particular component. We made this choice because, when we initially create parts in TeamConnection, we do not want to apply any process management. We can of course change the process later so that it reflects the state of our development project.

We also used a *naming convention* for this particular component. We decided to use a *prefix* for all child components that is based on the name of the project's *top* component. We used a three-letter prefix and it just happened that the *top* component's name was only three characters long. Figure 51 shows the component structure for our *AFT* project.

+-----+
|
|

PICTURE 53

Figure 51. Our Component Structure

With the help of Table 9 in topic A.0 in Appendix A, "Component Hierarchy Creation Aid" in topic A.0, we created Table 7 to aid us in the creation of our component structure.

Table 7. Component Hierarchy Creation Aid

Family Name		fam2008
Parent Name (-parent)	Child Name (-create -delete -link -recreate -unlink)	
AFT	aft_OS2	
AFT	aft_AIX	
AFT	aft_MVS	
aft_OS2	aft_OS2_C	
aft_AIX	aft_AIX_C	
aft_AIX	aft_AIX_KSH	
aft_MVS	aft_MVS_C	
aft_MVS	aft_MVS_ASM	
aft_MVS	aft_MVS_JCL	

Once our components are created, they can be displayed by using the **TeamConnection - Components** window (see Figure 52). In Figure 52, we use the tree view to display our components.

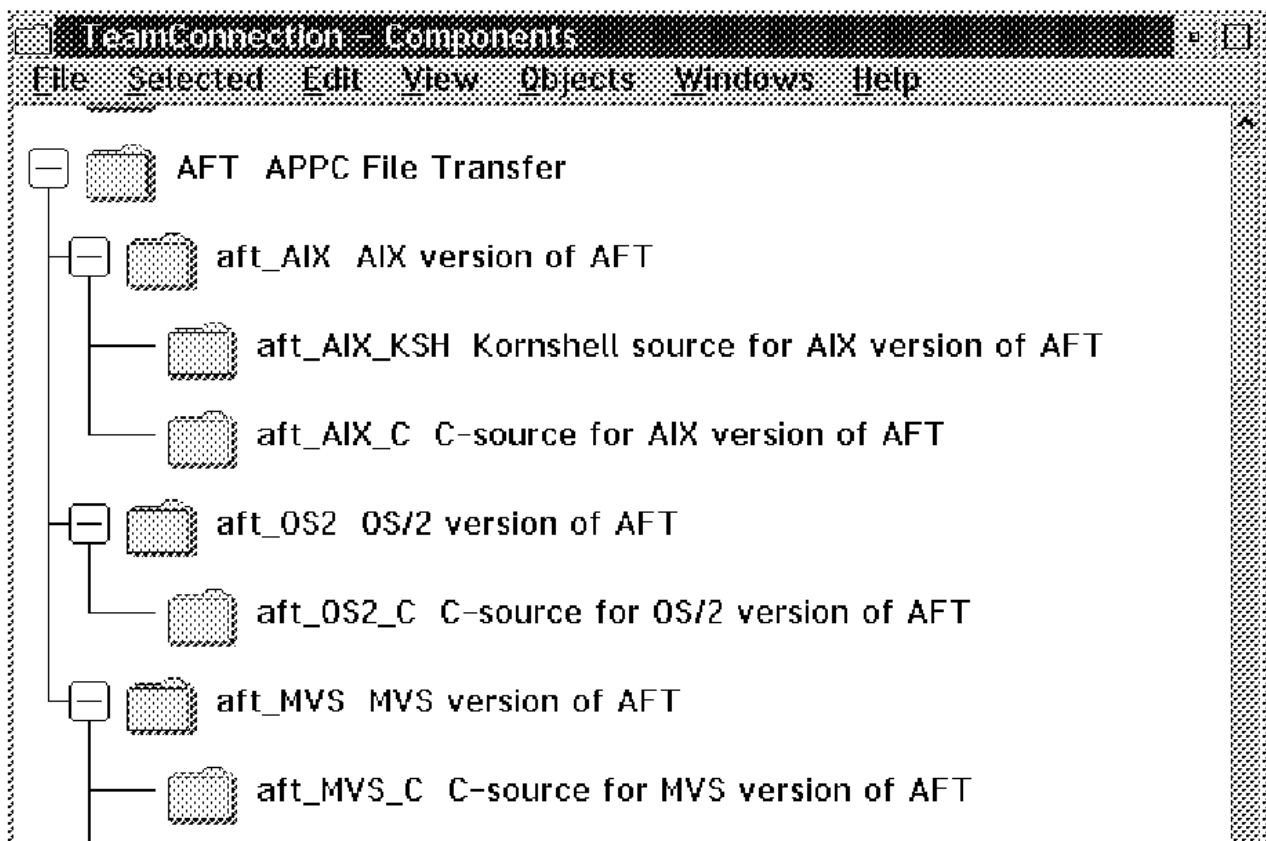


Figure 52. The Components Window

The following *pop-up actions* are available in the **TeamConnection - Components** window:

View

Displays the **View Component Information** window that enables you to see information about components

Process configurations

Displays the **View Component Process Configurations** window that enables you to see information about the processes that are defined for the specified components

Link

Displays the **Link Components** window that enables you to link components to a parent component

Unlink

Displays the **Unlink Components** window that enables you to detach components from a parent component

Re-create

Displays the **Re-create Components** window that enables you to reactivate components that were previously deleted

Delete

Displays the **Delete Components** window that enables you to delete components

Open defect

Displays the **Open Defect** window that enables you to open a defect

Open feature

Displays the **Open Feature** window that enables you to open a new feature

Add access

Displays the **Add Access** window that enables you to add an entry to an access list for a component

Add notification

Displays the **Add Notification** window that enables you to add an entry to a notification list for a component

Expand fully

Expands the selected component so that all descendant components are visible

Modify >

Select **Modify** to modify component names, owners and properties:

Name

Modifies the name of a component

Owner

Modifies the owner of components

Properties

Modifies the properties of components

Show >

In many cases, the information that TeamConnection displays when you select one of the options listed below depends on items you have selected in the corresponding panel. If you have not selected any objects in the window, TeamConnection cannot determine what you want to display. It therefore brings up the **Filter** window so that you can specify what you want displayed.

Access lists

Displays the access lists defined for the selected components

Notification lists

Displays the notification lists defined for the selected components

Defects >

Select **Defects** to view all of the defects assigned to the selected components or the defects assigned to the

selected components that are in a specific state.

All	Displays the defects assigned to the selected components
Open	Displays the defects in the open state assigned to the selected components
Design	Displays the defects in the design state assigned to the selected components
Size	Displays the defects in the size state assigned to the selected components
Review	Displays the defects in the review state assigned to the selected components
Working	Displays the defects in the working state assigned to the selected components
Verify	Displays the defects in the verify state assigned to the selected components
Returned	Displays the defects in the returned state assigned to the selected components
Canceled	Displays the defects in the canceled state assigned to the selected components
Closed	Displays the defects in the closed state assigned to the selected components

Features >

Select Features to view all of the Features assigned to the selected components or the features assigned to the selected components that are in a specific state.

All	Displays the features assigned to the selected components
Open	Displays the features in the open state assigned to the selected components
Design	Displays the features in the design state assigned to the selected components
Size	Displays the features in the size state assigned to the selected components
Review	Displays the features in the review state assigned to the selected components
Working	Displays the features in the working state assigned to the selected components
Verify	Displays the features in the verify state assigned to the selected components
Returned	Displays the features in the returned state assigned to the selected components
Canceled	Displays the canceled features assigned to the selected components

Closed

Displays the features in the closed state
assigned to the selected features

Parts

Displays the parts belonging to the selected
components

Releases

Displays the releases associated with the selected
components

When you access the pop-up choices for an item by clicking on an item using the right mouse button, you notice that the list varies depending on the item you select. TeamConnection lists only those options that are valid based on the item you selected and its current state.

Use the **Component Filter** window (see Figure 49) to search for components. The data you type in the filter fields is case sensitive; you must type it exactly as it was entered in the database.

6.3.2 Transfer Component Ownership

Use the **Modify Component Owner** window (see Figure 53) to assign a new owner to components. For example, Felix, a new member, joins your team. Part of Felix's responsibility is to take over writing the product help documentation. You use the **Modify Component Owner** window to reassign the documentation component to Felix.

Select **Selected > Modify > Owner...** or **Modify > Owner...** from the **TeamConnection - Components** window to get to the **Modify Component Owner** window (or select **Actions> Components > Modify > Owner...** from the **TeamConnection - Tasks** window). If you receive a message saying that you do not have the necessary authority, contact your family administrator and request CompModify authority. You can type more than one item in some of these fields. If you do so, separate the items with a space.

Many fields provide a pull-down menu from which you can select items for the field. This pull-down contains a list of up to the last 10 values you previously entered in this field. If the list does not contain the value you want to use, select the **Show all choices** push button at the bottom of the list for a list of all of the valid choices for this field. To access the pull-down menu, either select the down arrow at the right of the field or press F7.

Here is a description of the fields in the **Modify Component Owner** window:

Components

Type the names of the components that you want to assign to a new owner. If you selected a component, TeamConnection displays that component's name in this field. You can change this name.

New owner

Type the name of the new component owner. The owner has authority to create an access list and a notification list for that component.

The following push buttons are valid:

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

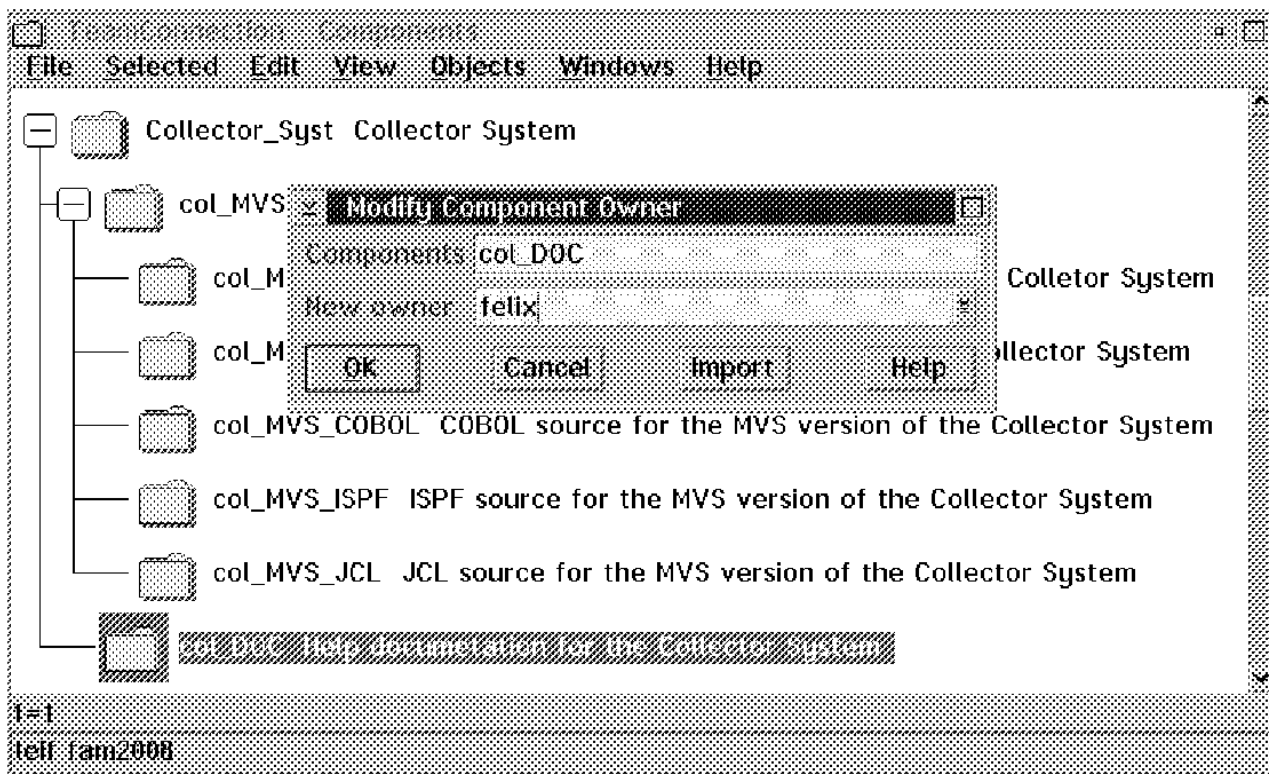


Figure 53. Modify Component Owner Window

6.3.3 Using the Command Line Interface

Use the **component** command to create and maintain a component structure for project control and management. The component structure or hierarchy consists of a top-level component called *root*. Every component below *root* is linked to one or more parent components and zero or more child components. Use *-link* and *-unlink* to redefine an existing component structure. You can create, delete, and recreate components, modify their properties, or view information about them.

Using the TeamConnection command line interface to create components is similar to using it to create users and host lists (see Figure 48 in topic 6.2.2). Figure 54 shows the commands used to create the components in the component tree shown in Figure 51 in topic 6.3.1.

```
+-----+
|
| teamc Component -create AFT -parent root -process prototype -owner leif
| teamc Component -create aft_OS2 -parent AFT -process prototype -owner leif
| teamc Component -create aft_OS2_C -parent aft_OS2 -process prototype -owner leif
| teamc Component -create aft_AIX -parent AFT -process prototype -owner leif
| teamc Component -create aft_AIX_C -parent aft_AIX -process prototype -owner leif
| teamc Component -create aft_AIX_KSH -parent aft_AIX -process prototype -owner leif
| teamc Component -create aft_MVS -parent AFT -process prototype -owner leif
| teamc Component -create aft_MVS_C -parent aft_MVS -process prototype -owner leif
| teamc Component -create aft_MVS_ASM -parent aft_MVS -process prototype -owner leif
| teamc Component -create aft_MVS_JCL -parent aft_MVS -process prototype -owner leif
|
+-----+
```

Figure 54. Creating Components through the Command Line Interface

Figure 55 shows an example of the commands used to create components with different component owners.

```
+-----+
|
| teamc Component -create toplevel -parent root -process maintenance -owner parkin
| teamc Component -create data -parent toplevel -process maintenance -owner parkin
| teamc Component -create require -parent toplevel -process maintenance -owner goti
| teamc Component -create code -parent toplevel -process maintenance -owner mccart
| teamc Component -create test -parent toplevel -process maintenance -owner orlo
|
+-----+
```

Figure 55. Creating Components with Different Owners

6.4 Creating a Release

A *release* is a user-defined TeamConnection object that contains all of the parts that must be built, tested, and distributed as a single entity.

Each TeamConnection part is managed by a component. An entire application is likely to contain parts from more than one component. Because you probably want to use the same parts in more than one version of an application, TeamConnection also groups parts into releases. A release is a way of identifying the exact version of all parts that belong to an application at a certain point in time.

Each part in TeamConnection is managed by at least one component and contained in at least one release. One release can contain parts from many components; a part can be included in several releases.

Each time a development cycle begins for the next version of an application, you can define a separate release. Each subsequent release of an application references many of the same parts as its predecessor. However, each release links to a particular version of individual parts. Thus maintenance of an older release can occur at the same time as development of a newer release.

Subtopics

6.4.1 Using the GUI

6.4.2 Using the Command Line Interface

6.4.1 Using the GUI

There are several ways in which you can access the **Create Releases** window (see Figure 57 on page 133). You can access it by:

- ☐ Selecting **Actions > Releases > Create...** from the **TeamConnection - Tasks** window (probably the easiest way to create your first release)
- ☐ Selecting **Selected > Create...** from the **TeamConnection - Releases** window

You can get to the **TeamConnection - Releases** window by selecting **Objects > Releases > Releases...** from either the **TeamConnection - Tasks** window or a task in your **TeamConnection - Tasks** window (if you created a task). If you select **Objects > Releases > Releases...** from the **TeamConnection - Tasks** window, you will first get the **Releases Filter** window (see Figure 56).

Use the **Release Filter** window to search for releases. The data you type in the filter fields is case sensitive; you must type it exactly as it was entered in the database. You can type more than one item in some of these fields. If you do so, separate the items with a space.

Here is an explanation of the fields in the **Release Filter** window:

Releases

Type the names of the releases you want to view.

Components

Type the names of the components associated with the releases. Every release must have a managing component. Through this component you can control access to the release and configuration notification according to each user's interest in release-related actions. A release has a one-to-one relationship with a component for management purposes and a one-to-many relationship with files for product-related activities.

Owner IDs

Type the user IDs of the owners of the releases.

Owner names

Type the names of the release owners.

Owner areas

Type the departments or areas where the owners work

Processes

Type the names of the processes that were defined for each release. You can use the down arrow to view and select from a list of available choices.

Add dates

Type the dates (yyyy/mm/dd) on which the releases were created.

Delete dates

Type the dates (yyyy/mm/dd) on which the releases were deleted.

Change dates

Type the dates (yyyy/mm/dd) on which the releases were last changed.

Descriptions

Type the information describing the purpose of the release.

For the following, select the radio buttons to indicate whether you want to list releases that have the characteristic or subprocess listed. For example, let's look at the track subprocess. If you select:

Yes

Lists only releases with the track subprocess

No

Lists only releases without the track subprocess

Both

Lists releases regardless of whether the track subprocess is defined

Use the previous example for each of the following:

Development modes

TeamConnection at Large Using the GUI

Select the appropriate push buttons to specify whether the release was created in serial or concurrent development mode.

Work area Subprocess

Specifies releases that have a work area subprocess. In the work area subprocess, all part changes must be made in reference to a work area.

Approval Subprocess

Specifies releases that have the approval subprocess. In the approval subprocess, part changes must be approved by everyone on an approver list.

Fix Subprocess

Specifies releases that have the fix subprocess. In the fix subprocess, users must mark fix records to indicate that part changes for a defect or feature are ready to be integrated with the rest of the parts in a release.

Driver Subprocess

Specifies releases that have the driver subprocess. The driver subprocess provides a method of integrating parts changed for defects and features with the rest of the parts in a release. It also provides a method of extracting the most recent versions of parts that will work together correctly.

Test Subprocess

Specifies releases that have the test subprocess. In the test subprocess, the users named in the environment list for a release must mark test records to record the results of testing changes in an environment.

Automatic pruning

Specifies releases that have the automatic pruning option turned on

Output versions

Type the version of the release where the outputs are saved.

History

A list of previous queries created by using this **Releases Filter** window

Query

The query you are about to submit

The screenshot displays the TeamConnection GUI's 'Releases Filter' window, which is used to define search criteria for releases. The main window is titled 'Releases Filter' and contains several sections:

- Releases:** A list of criteria with dropdown menus for selection and a 'No Sort' button. The criteria include: Releases (In), Components (In), Owner IDs (In), Owner names (In), Owner areas (in), Processes (in), Add dates (>), and Delete dates (>).
- Owner areas:** A section with dropdown menus for selection and a 'No Sort' button.
- Processes:** A section with dropdown menus for selection and a 'No Sort' button.
- Add dates:** A section with dropdown menus for selection and a 'No Sort' button.
- Delete dates:** A section with dropdown menus for selection and a 'No Sort' button.
- Change dates:** A section with dropdown menus for selection and a 'No Sort' button.
- Descriptions:** A section with dropdown menus for selection and a 'No Sort' button.
- Development modes:** A section with radio buttons for 'Concurrent' and 'Serial', and a 'Both' button.
- Work area Subprocess:** A section with radio buttons for 'Yes' and 'No', and a 'Both' button.
- Approval Subprocess:** A section with radio buttons for 'Yes' and 'No', and a 'Both' button.
- Fix Subprocess:** A section with radio buttons for 'Yes' and 'No', and a 'Both' button.
- Driver Subprocess:** A section with radio buttons for 'Yes' and 'No', and a 'Both' button.
- Test Subprocess:** A section with radio buttons for 'Yes' and 'No', and a 'Both' button.
- Automatic pruning:** A section with radio buttons for 'Yes' and 'No', and a 'Both' button.
- Output versions:** A section with dropdown menus for selection and a 'No Sort' button.

At the bottom of the window, there are buttons for 'OK', 'Apply', 'Clear', 'Save to Task List', 'Generate Query', 'Cancel', and 'Help'. The 'Generate Query' button is highlighted in red.

Figure 56. Releases Filter Window

Use the **Create Releases** (see Figure 57) window to create a new release of a product. You can also use this window to turn on concurrent development and automatic pruning. If you create a release, you become its owner, and you have authority to define an approver list and an environment list for that release. If you receive a message saying that you do not have the necessary authority, contact your family administrator and request ReleaseCreate authority.

The following fields are available in the **Create Releases** window:

Releases

Type the names of the releases you want to create. A release name cannot exceed 15 characters. Do not use the name of an existing release in your family or that of a release that has been removed from active status. Use valid OS/2, release, and driver member names. You can type up to 15 alphanumeric characters (a-z, A-Z, 0-9), plus ordinary punctuation. Do not use vertical bars, colons, ASCII control characters, shell metacharacters, or reserved device names such as KBD\$, PRN, NUL, COM1, COM2, or CLOCK\$. The following symbols cannot be used in file or directory names:

[0-1f]hex / : * ? < > | & - \$ \ " ' .

Component

Type the component to which the release belongs. The access list for the specified component determines who has authority to perform actions on the release.

Process

Select a process for the release. The following processes are supplied with TeamConnection:

- ☐ development
- ☐ emergency_fix
- ☐ maintenance
- ☐ no_track
- ☐ preship
- ☐ prototype
- ☐ test
- ☐ track_approval
- ☐ track_full
- ☐ track_driver
- ☐ track_only
- ☐ track_test

Your family administrator might have removed, added to, or changed these processes. TeamConnection help is not available for any changes that your family administrator makes.

Owner

Assign ownership of the release to someone other than yourself. If you do not specify another owner, you own the release. The owner has authority to define an approver list and an environment list for that release.

Description

Type a description of the purpose of the release. The maximum length is 63 characters. Do not use the dollar sign (\$), double or single quotes, or the back slash (\) character.

Environment

If the process you specify for the release includes the test subprocess, you can specify an environment to create an environment list. The environment can be anything you want to specify, such as an operating system or a hardware configuration. TeamConnection creates a test record for each environment you specify.

Tester

If the process you specify for the release includes the test subprocess, specify the user ID of a tester to create an environment list. The tester is responsible for testing changes in the environment you specified.

Approver

If the release includes the approval process, you can specify the user ID of an approver of the release. Specifying an approver creates an approver list for the release.

Database

Type the path and file name of the database used for storing parts for the release. If you do not specify a database, the parts are stored in the family database.

Maximum number of output versions

Type the maximum number of output versions.

Automatic version pruning

Turns on the automatic pruning function for the release

Concurrent development mode

Turns on concurrent development mode for the release

Figure 57. Create Releases Window

For the *Process*, we chose the *prototype* process (see Figure 57) for the same reason we chose the *prototype* process for the component process: we do not want to apply any process management at this point (before we have populated the release). We can of course change the process later, so that it reflects the state of our development project.

Once we have created one or more releases, we can use the **TeamConnection - Releases** window (see Figure 58) when we want to work with our releases. The following are actions that are accessible from the **Selected** menu item in the **TeamConnection - Releases** window:

Create

Displays the **Create Releases** window that enables you to create a

new release of a product

View

Displays the **View Release Information** window that enables you to see information about releases

Process configurations

Displays the **View Release Process Configurations** window that enables you to look at the process that was defined for each release

Extract

Displays the **Extract Releases** window that enables you to copy the parts from a version release

Link

Displays the **Link Releases** window that enables you to copy the parts in one release to a new release (the release to which you link)

Re-create

Displays the **Re-create Releases** window that enables you to make previously deleted releases active again

Prune

Displays the **Prune Release** window that enables you to delete a branch of versions from a release

Delete

Displays the **Delete Releases** window that enables you to delete releases

Add approvers

Displays the **Add Approvers** window that enables you to add an entry to an approver list for a release

Add environments

Displays the **Add Environments** window that enables you to add an entry to an environment list for a release

Create builders

Displays the **Create Builders** window that enables you to create builders

Create parsers

Displays the **Create Parsers** window that enables you to create parsers

Create work areas

Displays the **Create Work Areas** window that enables you to create work areas

Modify >

Select one of the following to modify the name, owner, component, or properties of a release:

Owner

Modifies the owner of a release

Component

Modifies the component of a release

Properties

Modifies the properties of a release

Show >

Select Show to display the builders, parsers, work areas, versions, locked parts, approver lists, environment lists, parts, and drivers. In many cases, the information that TeamConnection displays when you select one of the following options depends on items you have selected in the corresponding panel. If you have not selected any objects in the window, TeamConnection cannot determine what you want to display. It therefore brings up the **Releases Filter** window (see Figure 56) so that you can specify what you want displayed.

Builders

Displays the builders for the selected releases

Parts locked

Displays the parts locked for the selected releases

Approver lists

Displays the approver lists for the selected releases

Environment lists

Displays the environment lists for the selected releases

Parts

Displays the **TeamConnection - PartFull** window for the selected releases

Drivers

Displays the drivers for the selected releases

Parsers

Displays the parsers for the selected releases

Versions

Displays the versions for the selected releases

Work areas

Displays the **Work Area** window that enables you to work with work areas

All of the above items except **Create** are also available through the *pop-up* menu. When you access the pop-up choices for an item by clicking on an item with the right mouse button, you notice that the list varies depending on the item you select. TeamConnection lists only those options that are valid based on the item you selected and its current state.

TeamConnection - Releases					
File Selected Edit View Objects Windows Help					
Release	Component	Owner	Owner Name	Area	Process
AFT-MVS	AFT	leif	Leif Trulsson	BLD80/E2_RM412	prototype
Collector_AIX_1	aft_AIX	leif	Leif Trulsson	BLD80/E2_RM412	no_track
AFT-AIX	AFT	leif	Leif Trulsson	BLD80/E2_RM412	development
AFT-QS2	AFT	leif	Leif Trulsson	BLD80/E2_RM412	development
testrel	test	leif	Leif Trulsson	BLD80/E2_RM412	prototype
SG26-2008_R1	SG26-2008	leif	Leif Trulsson	BLD80/E2_RM412	prototype
Collector_MVS_1	col_MVS	leif	Leif Trulsson	BLD80/E2_RM412	no_track
1=1					
leif tam2008 0 of 7 selected					

Figure 58. TeamConnection - Releases Window

6.4.2 Using the Command Line Interface

Use the **release** command to:

- ☐ Create, modify, delete, and re-create releases
- ☐ Extract the set of parts associated with a release
- ☐ Link parts within releases with those in other releases
- ☐ View information about existing releases

A release is a set of parts that must be built, tested, and distributed as a whole. A release must be created in relation to a component that manages access and notification for the release. If you create a release, you become its owner, and you have implicit authority to define an approval list and an environment list for that release. Release names must be unique within a family.

When you create a release, you must choose a process if you use the **-process** flag. A process contains different combinations of TeamConnection subprocesses. TeamConnection subprocesses determine the states of the work areas within a release. For release processes, you can specify the track, approval, fix, driver, and test subprocesses. Processes are configured by your family administrator, who can modify current processes and define new processes. For a list of the valid release processes and the TeamConnection subprocesses they include, use the **report -view cfgrelproc** command.

When you create a release, you must choose between serial and concurrent development. This is the only time you can select the development type, because once set, you cannot change it. Choose concurrent development by using the **-concurrent** flag on the **release -create** command. Otherwise, the release defaults to serial development. For more information about the **release** command syntax, refer to the *IBM TeamConnection for OS/2 Commands Reference*, SC34-4501.

Figure 59 shows the command line syntax for creating releases.

```
+-----+
| teamc Release -create AFT-OS2 -component aft_OS2 -process prototype -wner leif ,
|               -environment os2 -tester stade6 -approver leif
| teamc Release -create AFT-MVS -component aft_MVS -process prototype -wner leif ,
|               -environment os2 -tester felix -approver leif
+-----+
```

Figure 59. Creating Releases through the Command Line Interface

6.5 Creating a Work Area

A work area is a TeamConnection object created to monitor the progress of changes within a release, resolve a specific defect, or implement a specific feature. As the work area is related to the specific changes to be made (not to specific users), it can be shared among multiple users.

A release is linked to a particular version of each of its objects (files or elements). As you check out objects, update them, and then check them back in, TeamConnection keeps track of all of the changes, even when more than one user updates the same object at the same time. To make this possible, TeamConnection uses something called a work area, a staging or development area in which you can:

- ☐ Check out objects (files and elements) from a release
- ☐ Update any or all of the objects without affecting the current version in the release
- ☐ Get the latest copies of the objects in the release, including any changes committed by other users
- ☐ Get the latest copies of the objects in another user's work area
- ☐ Freeze the work area, making a snapshot of the objects as they exist at a particular instant
- ☐ Commit the entire work area
- ☐ Do a build on the files in the work area

You create a work area and associate it with a release. At the time the work area is created, you see the most current view of the release and all the objects that it contains. You can check out the objects visible to the work area, make modifications, and check them back into the work area. You can specify applications within the work area to be built by the build tool. Finally, when you are satisfied that the work area contains a set of modifications that should be saved, you can request that TeamConnection freeze the work area. This makes a system version of all of the objects contained in the work area.

In the sections that follow, we show you how to create a work area by using either the GUI or the command line interface.

Subtopics

6.5.1 Using the GUI

6.5.2 Using the Command Line Interface

6.5.1 Using the GUI

By now you should be familiar with the various ways by which you can access windows within TeamConnection (through the **Actions** or **Options** menu items, or a task defined in the task list in the **TeamConnection - Tasks** window).

Use the **Create Work Areas** window (see Figure 60) to create a new work area to follow a defect or feature in a particular release. If sizing records exist for a defect or feature and the component includes the design, size, and review subprocess, TeamConnection automatically creates a work area when the defect or feature moves to the working state. If no sizing records exist, you must manually create a work area for each release that is using the track subprocess and is affected by a defect or feature. If an approver list exists for the release, TeamConnection creates the work area in the approve state. Otherwise, the work area is created in the fix state. The following conditions apply:

- ☐ The release must have the track subprocess specified.
- ☐ The defect or feature must be in the working or verify state.
- ☐ If the release includes the approval subprocess, TeamConnection automatically creates an approval record for each user on the approver list.
- ☐ If the release includes the fix subprocess, TeamConnection automatically creates fix records based on the sizing records for the defect or feature.
- ☐ If the release includes the test subprocess, TeamConnection automatically creates test records according to the entries in the environment list for the release.

If you receive a message saying that you do not have the necessary authority, contact your family administrator and request WorkAreaCreate authority.

You can type more than one item in some of the fields in **Create Work Areas** window. If you do so, separate the items with a space. Many fields provide a pull-down menu from which you can select items for the field. This pull-down contains a list of up to the last 10 values you previously entered in this field. If the list does not contain the value you want to use, select the **Show all choices** push button at the bottom of the list to get a list of all of the valid choices for this field. To access the pull-down menu, either select the down arrow at the right of the field or press F7.

The following is an explanation of the fields and buttons in the **Create Work Areas** window:

Work areas

Type the name of the work area with which you want to work.

Defects/Features

Type the names of the defects or features associated with the work area.

Releases

Type the names of the releases with which the work area is associated.

Target

Type the completion target. For example, you can use any of the following:

- ☐ The date when the work area is to be completed
- ☐ The driver in which the work area is to be included
- ☐ Any other goal or remark associated with the work area

Owner

Type the name of an owner other than yourself. TeamConnection assigns ownership of the work area to whomever you specify. If this field is blank, TeamConnection assigns you as the owner.

Select one of the following push buttons:

OK

Processes the information that you typed and closes the window

Apply

Processes the information that you typed and leaves the window open

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

The screenshot shows a 'Create Work Areas' dialog box. The fields are filled with the following data:

Field	Value
Work areas	aftmvs
Defects/Features	
Releases	AFT-MVS
Target	960531
Owner	leif

Buttons at the bottom: Ok, Apply, Cancel, Import, Help.

Figure 60. Create Work Areas Window

When you have created your work areas, you can then bring up the **TeamConnection - Work Areas** window (see Figure 62) by either selecting **Objects > Work Area > Work areas...** in the **TeamConnection - Tasks** window (or any other window containing the **Objects** menu item), or if you have create a task in your **TeamConnection - Tasks** window, by selecting that task. If you select **Objects > Work Area > Work areas...** from the **TeamConnection - Tasks** window, you will first get the **Work Area Filter** window (see Figure 61).

Use the **Work Area Filter** window to search for work areas. The data you type in the filter fields is case sensitive; you must type it exactly as it was entered in the database. You can type more than one item in some of these fields. If you do so, separate the items with a space. The following fields and buttons are defined in the **Work Area Filter** window:

Work areas

Type the name of the work area with which you want to work.

Releases

Type the names of releases in which work is being done for the work areas.

Defects/Features

Type the defects or features associated with the work areas.

References

Type the values assigned for defects or features that you use to group related defects or features.

States

Type the states for work areas. The following are valid states:

- ☐ approve
- ☐ fix
- ☐ integrate
- ☐ commit
- ☐ test
- ☐ complete

Targets

Type the completion targets. For example, you can use any of the following:

- ☐ The date when the work area is to be completed
- ☐ The driver in which the work area is to be included
- ☐ Any other goal or remark associated with the work area

Add dates

Type the dates (yyyy/mm/dd) on which the work areas were

created.

Owner IDs

Type the user IDs of the originators of the work areas.

Owner names

Type the names of the users who opened the work areas.

Owner areas

Type the departments or areas in which the originators work.

Actual

Type the indicators that specify when the work areas were actually completed. For example, you can use any of the following:

- ☐ The date when the work area is to be completed
- ☐ The driver in which the work area is to be included
- ☐ Any other goal or remark associated with the work area

Change dates

Type the dates (yyyy/mm/dd) on which the work areas or information associated with the work areas were last changed.

Prefixes

Type the values specified when the defects or features were opened. Your family administrator can configure these values. TeamConnection help is not available for any values or fields configured by your family administrator. You can use the down arrow to view and select from a list of available choices.

Abstracts

Type the summaries of the defects or features that the work areas address.

Types

Type the value that identifies the type of the work area, for example, defect or feature.

Branch point

Type the version of the release where the work area originated.

History

A list of previous queries created by using this **Work Area Filter** window

Query

The query you are about to submit

OK

Starts the query and closes the window

Apply

Starts the query but leaves the window open. This is useful if you want to test a query and then return to the **Work Area Filter** window to save the query to the **TeamConnection - Tasks** window.

Clear

Clears all the information that you typed and leaves the **Work Area Filter** window open

Save to Task List

Displays the **Edit Task List** window that enables you to add the query to the **TeamConnection- Tasks** window

Generate query

Displays the query in the query line that is generated from the data you typed in the filter fields

Cancel

Closes the window without starting a query

Help

Displays help information about the window

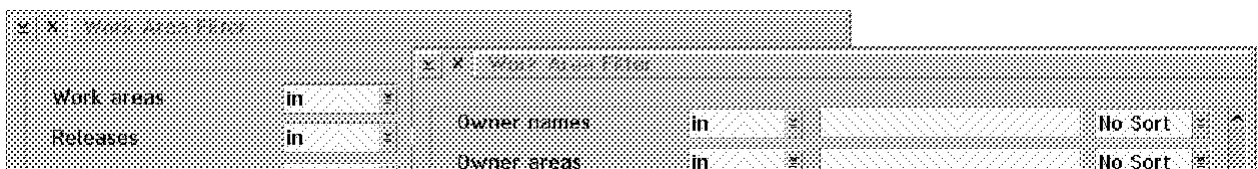


Figure 61. Work Area Filter Window

Use the **TeamConnection Work Areas** window (see Figure 62) to display the work areas associated with a defect or feature. A work area is a location where you put your work, and you use it to monitor a defect or feature in one release. Work areas are created automatically if the component for the release includes the design, size, and review subprocess and a sizing record for a defect or feature has been accepted.

When a work area is created automatically, the initial work area owner is the owner of the release that the work area references. You can also create work areas manually when a defect or feature is in the working state.

Because a single defect or feature can affect more than one release, you must create a separate work area for each release affected by the defect or feature. This defect or feature, together with the release, uniquely identifies a work area within a family.

By default, a work area is owned by its creator. Once you create a work area, it remains associated with the versions of the parts that were committed at that time. You can make changes, do builds, and debug in your work area without other changes in the release affecting the work you are doing.

When you are ready, you can refresh your work area and integrate any other changes (changes in the release that have occurred since you created your work area) with the changes you have made in your work area. In the case of collisions (which can happen only if you are working in concurrent development mode), you are presented with a list of conflicts to be resolved, and TeamConnection provides you with a tool to assist with the merging of the changes.

From the **Selected** menu item in the **TeamConnection - Work Areas** window (see Figure 62) you have the following selections:

Create work areas

Displays the **Create Work Areas** window that enables you to create new work area to follow a defect or feature in a particular release

View

Displays the **View Work Area Information** window that enables you to view information about a work area

Integrate

Displays the **Integrate Work Areas** window that enables you to move a work area from the fix state to the integrate state if no more part changes are to be made

Check

Displays the **Check Work Areas** window that enables you to determine the prerequisite and corequisite work areas for a particular work area

Commit

Displays the **Commit Work Areas** window that enables you to move a work area from the integrate state to the commit state

Test

Displays the **Test Work Areas** window that enables you to move a work area from the commit state to the test state. Part changes cannot be associated with the work area that you want to move.

Complete

Displays the **Complete Work Areas** window that enables you to move a work area from the test state to the complete state

Freeze

Displays the **Freeze Work Areas** window that enables you to prevent additional changes to a work area

Refresh

Displays the **Refresh Work Areas** window that enables you to update the parts in the work area with any changes from the release

Fix

Displays the **Fix Work Areas** window that enables you to move a work area from the integrate state back to the fix state when you have to make additional part changes

Cancel

Displays the **Cancel Work Areas** window that enables you to delete a work area

Create approval records

Displays the **Create Approval Records** window that enables you to create an approval record

Create fix records

Displays the **Create Fix Records** window that enables you to create a fix record

Add driver members

Displays the **Add Driver Members** window that enables you to create work areas as members of a specific driver

Add corequisites

Displays the **Add Corequisite Work Areas** window that enables you to add work areas to an existing group of corequisite work areas or create a group of corequisite work areas

Remove driver members

Displays the **Remove Driver Members** window that enables you to remove work areas as members of a specific driver

Remove corequisites

Displays the **Remove Corequisite Work Areas** window that enables you to remove work areas from a group of corequisite work areas

Modify >

Select one of the following to change the owner or the properties of a work area:

Owner

Modifies the owner of work areas

Properties

Modifies the properties of work areas

Show >

Select Show to display the note history, change history, approval records, fix records, test records, or driver members. In many cases, the information that TeamConnection displays when you select one of the options listed below depends on items you have selected in the corresponding panel. If you have not selected any objects in the window, TeamConnection cannot determine what you want to display; it therefore brings up the **Filter** window for the selected choice so that you can specify what you want displayed.

Collision records >

Displays one of the following associated with a work area:

All

Displays all collision records

Active

Displays all of the active collision records

Accepted

Displays all of the accepted collision records

Reconciled

Displays all of the reconciled collision records

Rejected

Displays all of the rejected collision records

Parts

Displays the parts associated with a work area

Versions

Displays the versions associated with a work area

Note history

Displays the note history information for the selected item

Defects/Features

Displays the defects or features associated with the work area. If no defects or features are associated with the work area, TeamConnection displays the defects filter.

Change history

Displays the change history information for releases, drivers, and parts

Approval records

Displays the approval records for the selected work areas

Fix records

Displays the fix records for the selected work areas

Test records

Displays the test records for the selected work areas

Driver members

Displays the driver members for the selected work areas

All of the above items except **Create** are also available through the *pop-up* menu. When you access the pop-up choices for an item by selecting an item with the right mouse button, you notice that the list varies depending on the item you select. TeamConnection lists only those options that are valid based on the item you selected and its current state.

TeamConnection - Work Areas					
File Selected Edit View Objects Windows Help					
Name	State	Release	Branch Point	Defect Name	Reference
lt_collect_mvs	fix	Collector_MVS_1	Collector_MVS_1		
testwa	fix	Collector_MVS_1	Collector_MVS_1		
testwa	fix	testrel	testrel		
imp-scr	fix	testrel	testrel		
w2008	fix	SG26-2008_R1	SG26-2008_R1		
imp	fix	Collector_MVS_1	Collector_MVS_1		
1 of 7 selected					

Figure 62. TeamConnection - Work Areas Window

6.5.2 Using the Command Line Interface

Use the **workarea** command (see Figure 63) to create, modify, reassign, delete, freeze, refresh, and view information about a work area and to change the state of a work area. The states through which a work area moves depend on the TeamConnection subprocesses included in the associated release process. A release process can include the track, approval, fix, driver, or test subprocesses, or none at all.

If the track process is turned on for the release, a work area must be associated with a defect or feature. The default name for these work areas is the name of the defect or feature.

You can also create a work area with a specified name. If the release does not have the track process, you must create the work area with a specified name. The user who creates the work area becomes the owner of the work area unless a different owner is specified when the work area is created.

```
+-----+
|
| teamc WorkArea -create -name aftmvs -release AFT-MVS -owner leif
| teamc WorkArea -create -name w2008 -release SG26-2008_R1 -owner leif
|
+-----+
```

Figure 63. Creating Work Areas through the Command Line Interface

6.6 Creating Parts

A part is a collection of data stored by the TeamConnection server and retrieved by a path name. Any text or binary part used in a development project can be created as a TeamConnection part. Examples include source code, executable programs, documentation, and test cases.

Some common actions performed with parts include:

Create

Store a part from a workstation on the server; from then on, TeamConnection keeps track of all changes made to the part.

Check out

Get a copy of the part to make changes to it.

Check in

Put the part back into the library.

Extract

Get a copy of the part without planning to make changes to it.

Build

Apply a specified translation process to source parts and create (target) parts.

A TeamConnection part is uniquely identified by its path name and the name of the release in which it is contained. Both the release name and the path name must be specified whenever you perform a TeamConnection action on a part. Multiple releases can share the same part.

Any part that is contained in two or more releases is a shared part. A shared part consists of two TeamConnection parts sharing information. Each paired path name and release represents a separate TeamConnection part.

If the developers of one release need to use a part that is already contained in another release, they can link that part to their release (if they have the proper authority). Both releases would then have a link to the current version of the part.

Subtopics

6.6.1 Common Part

6.6.2 Using the GUI

6.6.3 Using the Command Line Interface

6.6.1 Common Part

A common part is a part shared among multiple releases where each release references the same current version of the part. Checking out a common part from any release locks the current version of the part in all of the releases. Because both releases are linked to the same current version, the part is locked in both releases.

In releases that include the track subprocess, common parts can be maintained automatically. To check in a common part to the server and maintain the common link, specify the release from which you checked out the part and any other releases in which it is common. In this way any changes made to a common part can be reflected in more than one release through a single check-in action.

In releases without the track subprocess, the common link is broken automatically as soon as the part is checked in.

When development begins a new release of a product, all parts in the current release can be linked to a new release so that, initially, all of the parts are common to both releases. As development of the releases progresses, the common link between the parts can be broken to separate development of the new release from maintenance of the current release.

In the sections that follow we take you through the creation of parts by using both the GUI and the command line interface.

6.6.2 Using the GUI

As you probably are familiar with by now, there are various ways that you can access windows within TeamConnection (through the **Actions** or **Options** menu items, or a task defined in the task list in the **TeamConnection - Tasks** window). For parts, there is yet another way, through the **TeamConnection - Work Areas** window. By selecting **Show > Parts** in the **TeamConnection - Work Areas** window, you will get to the **TeamConnection - Parts** window. But the first time you create a part you will probably use **Actions > Parts > Create...** from the **TeamConnection - Tasks** window.

Use the **Create Parts** window (see Figure 64) to do the following:

- ☐ Create new parts on the server. Part names created on the server are case sensitive; they must be retrieved using the same case in which they were created.
- ☐ Establish relationships between parts so that you can use the build function.

All parts you create using the **Create Parts** window have a part type of *file*. If you want to create a part with a different part type, you must use the **part -create** command. Refer to the *IBM TeamConnection for OS/2 Commands Reference*, SC34-4501, for additional information about and the syntax of the TeamConnection commands. You must have a work area to check in or create parts.

You can type more than one item in some of the fields in the **Create Parts** window. If you do so, separate the items with a space. Many fields provide a pull-down menu from which you can select items for the field. This pull-down contains a list of up to the last 10 values you previously entered in this field. If the list does not contain the value you want to use, select the **Show all choices** push button at the bottom of the list to get a list of all of the valid choices for this field. To access the pull-down menu, either select the down arrow at the right of the field or press F7.

The following is an explanation of the fields and buttons in the **Create Parts** window:

Part names

Type the names of the parts you want to create on the server. These names can include both the directory name and the base part name. For example, you could create parts called *DOC\HELP.SCRIPT* and *DOC\READ.SCRIPT* in this field. If you then extracted or checked out these parts, the base parts would be placed in a *DOC* directory on your workstation. If you had specified *C:\TEAM* as the default directory in your Settings notebook, the parts would appear on your workstation as follows:

```
C:\TEAM\DOC\HELP.SCRIPT
C:\TEAM\DOC\READ.SCRIPT
```

Note: If you do not use a high performance file system (HPFS), TeamConnection truncates the file extension to three characters.

The part names must be unique within the context of a release and a family.

Release

Type the name of the release to which you want to assign the parts you are creating.

Work area

Type the name of the work area to which you want to assign the parts you are creating.

Component

Type the name of the component to which you want to assign the parts you are creating.

File type

Select one of the following radio buttons to specify the file types of the parts you are creating:

Text

Specifies a part with ASCII- or EBCDIC-encoded data.

Binary

Specifies a part other than ASCII- or EBCDIC-encoded data. Examples of such parts are graphical, object code, or executable parts. To create a placeholder part, select **Binary**.

None

Specifies that there is no part associated with this object.

Source

Select one of the following radio buttons to specify the location from which you are getting the contents of the part you are creating in TeamConnection. You can select one of the following:

Same

Specifies that the contents of the part you are creating are from an identically named part on your workstation

No source

Specifies that the part you are creating has no source and thus contains no data; it is a 0-byte part. To create a placeholder part on the server, select **No source**.

Copy from

Specifies that the contents of the part you are creating are from a part on your workstation with a different name. You must type the name of this part in the Source part field.

Source file

Type the name of the file on your workstation whose contents you want to use as the source for the new part you are creating.

Source directory

Type the directory where the source file is currently located.

Remarks

Type comments in this field to add comments to parts. The information you type is visible when you select a part and then select **View information** from the **Selected** menu. You can also select the **Edit** push button to type lengthy remarks or insert text from a file.

Parent

Type the parent that you want to specify for the part for use with the build function.

Parent type

Type the part type of the parent.

Relation to parent

Type the relationship type of the part. Select one of the following radio buttons to specify the relationship:

Input

Type the name of a part used as direct input to your build. An example would be a C language source part. If you start a build and this part has changed, the changed part will be part of the new build.

Output

Type the name of a generated output from a build, such as an OBJ or EXE part, or a part with no contents that serves as an organizer object. If you start a build and this part has changed, the changed part will be included in the new build.

Dependent

Type the name of a part that is needed for the build operation to complete but is not passed directly to the compiler. An example would be an include part. If you start a build and this part has changed, the changed part will be included in the new build.

Builder

Type the name of the builder you want to use when you use the build function.

Parameters

Type the parameters associated with the build.

Parser

Type the name of the parser you want to use when you use the

build function.

Temporary file

Select this check box to identify that the part is a temporary file.

Choices

Displays the **File Choices** window, which displays a list of the parts in the source directory you specified on the **Create Parts** window. From this window you can select the parts you want to appear in the Part names field.

Select

Displays the **Select Source File** window. From this window, you can select a part name from your workstation. The part you select appears in the Source file field.

OK

Processes the information that you typed and closes the window

Apply

Processes the information that you typed and leaves the window open

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

Create Parts		<input type="checkbox"/>
Part names	2008USE	Choices...
Release	SG26-2008_R1	≡
Work area	w2008	≡
Component	2008_PR5	≡
File type	<input checked="" type="radio"/> Text <input checked="" type="radio"/> Binary <input type="radio"/> None	
Source	<input checked="" type="radio"/> Same <input type="radio"/> No source <input checked="" type="radio"/> Copy from	
Source file	M:\2008-prs\2008USE.PRS	Select...
Source directory		
Remarks	Freelance Graphics file for the chapter "Using TeamConnection"	
<input type="button" value="Edit..."/>		
Parent		
Parent type		
Relation to parent	<input checked="" type="radio"/> Input <input type="radio"/> Output <input type="radio"/> Dependent	
Builder		
Parameters		
Parser		
<input type="checkbox"/> Temporary file		
<input type="button" value="OK"/> <input type="button" value="Apply"/> <input type="button" value="Cancel"/> <input type="button" value="Import"/> <input type="button" value="Help"/>		

Figure 64. Create Parts Window

6.6.3 Using the Command Line Interface

If you have to create multiple parts at the same time, as when you are initially populating the family with existing parts, using the command line interface is definitely the best choice.

Use the **part** command to put files or parts into the TeamConnection product and to work with individual files afterward. For example, you can store VisualGen parts in the TeamConnection product. You put a part into the TeamConnection product, using **part -create**. When you create a part, you can do any of the following:

- Take an existing file that is on your workstation and place it in the TeamConnection product.
- Create a placeholder part that will not have any content until a successful build of its input, using the *-empty* flag. Placeholders are used in building an application.
- Create a part that acts as a collector object so that you can synchronize the build of unrelated parts. For more information, refer to the *IBM TeamConnection for OS/2 User's Guide*, SC34-4499. These parts do not have bulk contents.

When you create a part in the TeamConnection product, you associate it with a release (to relate the part to a development effort) and a component (to control the ownership of and access to a part). You have the option of specifying a file mode when creating a part, using the *-fmode* flag. (For example, in OS/2, you can use the *-fmode* flag to specify that a part is read-only.) If a file mode is not specified, the current file mode is used. After a part is successfully created on the server, TeamConnection modifies the file attributes of the working copy of the part left on the client to read-only. All subsequent part access commands must specify the part name, release name, and possibly a work area name to identify the correct part.

You can have two parts with the same base name, as long as the *-type* is different for each part. Unless you specify a type on a part, all commands that reference parts default to file. Once the type is specified, it cannot be changed.

Parts that have a unique base name or path name within a release can be specified by their base name or path name; other parts must be specified by their full path name when performing TeamConnection actions against the part.

Figure 65 shows an example of creating parts by using the command line interface.

```
+-----+
|
| teamc Part -create ibmoupd.cob -release Collector_MVS_1
|           -component col_MVS_COBOL
|           -workarea imp -text -from D:\collect\source\ibmoupd1.cob
| teamc Part -create ibmbse1.cob -release Collector_MVS_1
|           -component col_MVS_COBOL
|           -workarea imp -text -from D:\collect\source\ibmbse3.cob
| teamc Part -create ibmbupd.cob -release Collector_MVS_1
|           -component col_MVS_COBOL
|           -workarea imp -text -from D:\collect\source\ibmbupda.cob
| teamc Part -create ibmbuenr.cob -release Collector_MVS_1
|           -component col_MVS_COBOL
|           -workarea imp -text -relative D:\collect\source
| teamc Part -create ibmbuins.cob -release Collector_MVS_1
|           -component col_MVS_COBOL
|           -workarea imp -text -relative D:\collect\source
| teamc Part -create ibmcust.asm -release Collector_MVS_1
|           -component col_MVS_ASM
|           -workarea imp -text -relative D:\collect\source
| teamc Part -create ibmdate.asm -release Collector_MVS_1
|           -component col_MVS_ASM
|           -workarea imp -text -relative D:\collect\source
| teamc Part -create ibmou001.pan -release Collector_MVS_1
|           -component col_MVS_ISPF
|           -workarea imp -text -relative D:\collect\source
| teamc Part -create ibmou002.pan -release Collector_MVS_1
|           -component col_MVS_ISPF
|           -workarea imp -text -relative D:\collect\source
| teamc Part -create ibmou003.pan -release Collector_MVS_1
|           -component col_MVS_ISPF
|           -workarea imp -text -relative D:\collect\source
| teamc Part -create ibmou004.pan -release Collector_MVS_1
|           -component col_MVS_ISPF
|           -workarea imp -text -relative D:\collect\source
| teamc Part -create ibmou005.pan -release Collector_MVS_1
|
+-----+
```

```

teamc Part -component col_MVS_ISPF
           -workarea imp -text -relative D:\collect\source
           -create ibmou006.pan -release Collector_MVS_1
           -component col_MVS_ISPF
           -workarea imp -text -relative D:\collect\source
teamc Part -create ibmoupd.cls -release Collector_MVS_1
           -component col_MVS_CLIST
           -workarea imp -text -from D:\collect\source\ibmoupd1.cls
teamc Part -create ibmbse1.jcl -release Collector_MVS_1
           -component col_MVS_JCL
           -workarea imp -text -from D:\collect\source\ibmbse3.jcl
teamc Part -create ibmbupd.jcl -release Collector_MVS_1
           -component col_MVS_JCL
           -workarea imp -text -from D:\collect\source\ibmbupda.jcl

```

Figure 65. Creating Parts through the Command Line Interface

Notice that we use the **-text** attribute flag to tell TeamConnection that the parts we are creating are text files (ASCII- or EBCDIC-encoded data). If we had created parts where the source were binary files, we would have used the **-binary** attribute flag to indicate this. The default attribute flag is **-text**.

It is worth noting the use of the **-from** and **-relative** attribute flags:

Attribute Flag and Argument	Description
-from <i>FileSpec</i>	Indicates the location of file contents
-relative <i>Name</i>	Creates, checks in, checks out, or extracts the specified part relative to the directory specified according to the complete path name of the part. Directories are created if needed. Extracting or checking out in order to copy the part by its full path name.

By using the **-from** attribute flag, we can create a part in TeamConnection that has a different name from the source; for example:

```

teamc Part -create ibmbupd.jcl -release Collector_MVS_1
           -component col_MVS_JCL
           -workarea imp -text -from D:\collect\source\ibmbupda.jcl

```

Here we take the file *D:\collect\source\ibmbupda.jcl* and create it as a part in TeamConnection with the name of *ibmbupd.jcl*.

When we use the **-relative** attribute flag, we tell TeamConnection to create the part with a specific name (as indicated by the **-create** attribute parameter), and the source can be found in the directory indicated by the **-relative** attribute parameter; for example:

```

teamc Part -create ibmbuenr.cob -release Collector_MVS_1
           -component col_MVS_COBOL
           -workarea imp -text -relative D:\collect\source

```

In this case, we create the *ibmbuenr.cob* part in TeamConnection using the same name as the file that is located in the *D:\collect\source* directory. If there is no file in *D:\collect\source* with the name of *ibmbuenr.cob*, TeamConnection will return an error, and the **part -create** will fail.

6.7 Working with Parts

In this section we take you through some of the basic things you need to know about working with parts. In the previous sections we separated the way you do things with the GUI and the way you do things with the command line interface. From now on, we intermix the two ways.

In "Common Part" in topic 6.6.1 we show you how to create parts using the **Create Parts** window. Once the parts have been created, you use the **TeamConnection - Parts** window (see Figure 66) or the **TeamConnection - PartFull** window (see Figure 67) to access and work with parts.

Use the **Part** window to access the versions of the parts that are associated with a specific work area (the **PartFull** window accesses all parts in the family, independent of the work area, version, driver, or release to which they belong). Use the **Parts** window to accomplish the following:

- ☐ Work with parts in a work area
- ☐ Do work area actions on the work area in which parts are located
- ☐ Perform other parts-related operations, such as build, connect, disconnect, or touch, on selected parts

In addition, you can do all of the regular part actions that are available from the **PartFull** window. All part actions apply to the nonfile parts, except for Check out, Check in, Extract, and Edit (from the GUI). *Nonfile parts* are parts that cannot be stored on the workstation as either a binary or text file. All part actions apply to file parts.

The following actions are available from the **Selected** menu item in the **TeamConnection - Parts** window:

Create

Displays the **Create Parts** window that enables you to create new parts on the server. You can also use this window to create build relationships between parts.

Work area >

Select Work area to view information about a work area; freeze, refresh, or commit parts in your work area; or delete a work area.

View

Displays the **View Work Area Information** window that enables you to view information about a work area

Integrate

Displays the **Integrate Work Areas** window that enables you to move a work area from the fix state to the integrate state if no more part changes are to be made

Check

Displays the **Check Work Areas** window that enables you to determine the prerequisite and corequisite work areas for a particular work area

Commit

Displays the **Commit Work Areas** window that enables you to move a work area from the integrate state to the commit state

Test

Displays the **Test Work Area** window that enables you to move a work area from the commit state to the test state. Part changes cannot be associated with the work area that you want to move.

Complete

Displays the **Complete Work Areas** window that enables you to move a work area from the test state to the complete state

Freeze

Displays the **Freeze Work Areas** window that enables you to prevent additional changes to a work area

Refresh

Displays the **Refresh Work Areas** window that enables you to update the parts in the work area with any changes from the release

Fix

Displays the **Fix Work Areas** window that enables you to move a work area from the integrate state back to the

fix state when you have to make additional part changes

Cancel

Displays the **Cancel Work Areas** window that enables you to delete a work area

Create approval records

Displays the **Create Approval Records** window that enables you to create an approval record

Create fix records

Displays the **Create Fix Records** window that enables you to create a fix record

Add driver members

Displays the **Add Driver Members** window that enables you to create work areas as members of a specific driver

Add corequisites

Displays the **Add Corequisite Work Areas** window that enables you to add work areas to an existing group of corequisite work areas or create a group of corequisite work areas

Remove driver members

Displays the **Remove Driver Members** window that enables you to remove work areas as members of a specific driver

Remove corequisites

Displays the **Remove Corequisite Work Areas** window that enables you to remove work areas from a group of corequisite work areas

Modify >

Select one of the following to change the owner or the properties of a work area:

Owner

Modifies the owner of work areas

Properties

Modifies the properties of work areas

Show >

Select Show to display the note history, change history, approval records, fix records, test records, or driver members. In many cases, the information that TeamConnection displays when you select one of the options listed below depends on items you have selected in the corresponding panel. If you have not selected any objects in the window, TeamConnection cannot determine what you want to display; it therefore brings up the **Filter** window for the selected choice so that you can specify what you want displayed.

Collision records >

Displays one of the following associated with a work area:

All

Displays all collision records

Active

Displays all of the active collision records

Accepted

Displays all of the accepted collision records

Reconciled

Displays all of the reconciled collision records

Rejected

Displays all of the rejected collision records

Parts	Displays the parts associated with a work area
Versions	Displays the versions associated with a work area
Note history	Displays the note history information for the selected item
Defects/Features	Displays the defects or features associated with the work area. If no defects or features are associated with the work area, TeamConnection displays the defects filter.
Change history	Displays the change history information for releases, drivers, and parts
Approval records	Displays the approval records for the selected work areas
Fix records	Displays the fix records for the selected work areas
Test records	Displays the test records for the selected work areas
Driver members	Displays the driver members for the selected work areas

View >

Enables you to view file contents, file information, and the build tree structure or to work with a file

View contents

Displays the **View Part Contents** window that enables you to view the contents of text parts

View information

Displays the **View Part Information** window that enables you to view information about parts

View build message

Displays the **View Build Message** window that enables you to view the output from the last build operation performed on an object

View build tree

Displays the **BuildView** window that enables you to see files in a build tree structure

Edit

Displays the **Edit Part** window that enables you to edit a part that is on either your workstation or the server.

Check out

Displays the **Check Out Parts** window that enables you to check out parts from the server

Check in

Displays the **Check In Parts** window that enables you to check in parts

Extract

Displays the **Extract Parts** window that enables you to copy TeamConnection parts to a disk

Lock

Displays the **Lock Parts** window that enables you to prevent other users from checking out the part

Unlock

Displays the **Unlock Parts** window that enables you to unlock a

part that is checked out (or has been previously locked) so that other users can check out the part

Build

Displays the **Build Parts** window that enables you to perform a build on selected parts

Cancel build

Displays the **Cancel Build** window that enables you to interrupt a build that is in progress

Connect

Displays the **Connect Parts** window that enables you to connect an existing build object as a child to another build object

Disconnect

Displays the **Disconnect Parts** window that enables you to remove the connection between two connected parts

Touch

Displays the **Touch Parts** window that enables you to change the date of a part to the current date

Link

Displays the **Link Parts** window that enables you to link parts (to create common or shared parts) from one release to another release (specified as the new release)

Undo

Displays the **Undo Parts** window that enables you to undo the most recent, uncommitted action that changed the specified parts

Re -create

Displays the **Re-create Parts** window that enables you to re-create previously deleted parts

Delete

Displays the **Delete Parts** window that enables you to delete the specified parts

Destroy

Displays the **Destroy Parts** window that enables you to completely remove the specified parts

Modify >

Select Modify to change the part path name, the part component, or the part properties:

Path name

Displays the **Modify Part Path Name** window that enables you to modify the path name of the specified part

Component

Displays the **Modify Part Component** window that enables you to specify a different component to manage your parts

Properties

Displays the **Modify Part Properties** window that enables you to change the properties of a part

Show >

Select Show to view the differences or change history. In many cases, the information that TeamConnection displays when you select one of the options listed below depends on items you have selected in the corresponding panel. If you have not selected any objects in the window, TeamConnection cannot determine what you want to display; it therefore brings up the **Filter** window for the selected choice so that you can specify what you want displayed.

Differences

Displays the differences between two parts

Change history

Displays the **Part Change History** window that enables you to display a graphical view of the objects in which parts have changed over time

Figure 66. TeamConnection - Parts Window

Use the **PartFull** window (see Figure 67) to access all versions of parts (the **Parts** window only accesses those parts that are related to a specific work area). You can access parts that belong to work areas, drivers, versions, or releases. The **PartFull** window also shows you in which work area and by whom a specific part is locked.

A TeamConnection part is any part that is under the control of a TeamConnection server. Each part is uniquely identified by its path name and the name of the release in which it is contained. All parts under TeamConnection control are managed by components.

When you create a part, you put the part under TeamConnection control. After a part is created, a copy of the part resides in the part system on the server. Changes to this part are made by checking it out to the client, making the changes, and then checking the changed copy back into the server. The components control the access to all parts under TeamConnection control and the releases group the parts for product-related activities. TeamConnection stores additional information about the part each time an action is performed against it.

The following actions are available from the **Selected** menu item in the **TeamConnection - PartFull** window (see Figure 67):

Create

Displays the **Create Parts** window that enables you to create new parts on the server. You can also use this window to create build relationships between parts.

View contents

Displays the **View Part Contents** window that enables you to view the contents of text parts

View information

Displays the **View Part Information** window that enables you to view information about parts

Edit

Displays the **Edit Part** window that enables you to edit a part that is on either your workstation or the server

Check out

Displays the **Check Out Parts** window that enables you to check out parts from the server

Check in

Displays the **Check In Parts** window that enables you to check in parts

Extract

Displays the **Extract Parts** window that enables you to copy TeamConnection parts to a disk

Lock

Displays the **Lock Parts** window that enables you to prevent other users from checking out the part.

Unlock

Displays the **Unlock Parts** window that enables you to unlock a part that is checked out (or has been previously locked) so that other users can check out the part

Link

Displays the **Link Parts** window that enables you to link parts (to create common or shared parts) from one release to another release (specified as the new release)

Undo

Displays the **Undo Parts** window that enables you to undo the most recent, uncommitted action that changed the specified parts

Re-create

Displays the **Re-create Parts** window that enables you to re-create previously deleted parts

Delete

Displays the **Delete Parts** window that enables you to delete the specified parts

Destroy

Displays the **Destroy Parts** window that enables you to completely

remove the specified parts

Modify >

Select **Modify** to change the part path name, the part component, or the part properties:

Path name

Displays the **Modify Part Path Name** window that enables you to modify the path name of the specified part

Component

Displays the **Modify Part Component** window that enables you to specify a different component to manage your parts

Properties

Displays the **Modify Part Properties** window that enables you to change the properties of a part

Show >

Select **Show** to view the differences or change history. In many cases, the information that TeamConnection displays when you select one of the options listed below depends on items you have selected in the corresponding panel. If you have not selected any objects in the window, TeamConnection cannot determine what you want to display; it therefore brings up the **Filter** window for the selected choice so that you can specify what you want displayed.

Differences

Displays the differences between two parts

Change history

Displays the **Part Change History** window that enables you to display a graphical view of the objects in which parts have changed over time

TeamConnection - PartFull				
File Selected Edit View Objects Windows Help				
New Path Name	Base Name	Release	Component	Work Area
2008PTC.PRS	2008PTC.PRS	SG26-2008_R1	2008_PRS	w2008
2008TERM.PRS	2008TERM.PRS	SG26-2008_R1	2008_PRS	w2008
2008PTC.PRS	2008PTC.PRS	SG26-2008_R1	2008_PRS	w2008
AFTARG.c	AFTARG.c	AFT-MVS	aft_MVS	aftmvs
AFTARG.obj	AFTARG.obj	AFT-MVS	aft_MVS	aftmvs
AFTDEFSH.h	AFTDEFSH.h	AFT-MVS	aft_MVS	aftmvs
AFTFILIO.c	AFTFILIO.c	AFT-MVS	aft_MVS	aftmvs
AFTFILIO.obj	AFTFILIO.obj	AFT-MVS	aft_MVS	aftmvs
AFTMVS.c	AFTMVS.c	AFT-MVS	aft_MVS	aftmvs
AFTMVS.load	AFTMVS.load	AFT-MVS	aft_MVS	aftmvs
AFTMVS.obj	AFTMVS.obj	AFT-MVS	aft_MVS	aftmvs
AFTMVSR.jcl	AFTMVSR.jcl	AFT-MVS	aft_MVS	aftmvs
AFTMVSS.jcl	AFTMVSS.jcl	AFT-MVS	aft_MVS	aftmvs
AFTSNA.c	AFTSNA.c	AFT-MVS	aft_MVS	aftmvs
AFTSNA.obj	AFTSNA.obj	AFT-MVS	aft_MVS	aftmvs
AFTSNA1.c	AFTSNA1.c	AFT-MVS	aft_MVS	aftmvs
baseName like 'X' order by baseName				
left fam2008				1 of 71 selected

Figure 67. The TeamConnection - PartFull Window

The actions or choices that we have just described are also available through *pop-up* choices. When you access the pop-up menu for an item by selecting an item using the right mouse button, you will notice that the list varies depending on the item you select. TeamConnection lists only those options that are valid based on the item you selected and its current state.

Remember that anything that you can do with the GUI you can also do with the command line interface. In the sections that follow we take a look at how to:

- ☐ **Check out** a part
- ☐ **Edit** a part
- ☐ **Check in** a part
- ☐ **Lock** a part
- ☐ **Unlock** a part
- ☐ **Extract** a part

Subtopics

- 6.7.1 Check Out a Part
- 6.7.2 Edit a Part
- 6.7.3 Check in a Part
- 6.7.4 Lock a Part
- 6.7.5 Unlock a Part
- 6.7.6 Extract a Part

6.7.1 Check Out a Part

You **check out** a part when you want to work with that particular part. When you check out the part, the part is retrieved from TeamConnection and stored in a directory on your local workstation. If you are using *serial development* for your release, TeamConnection will lock that particular version of the part in that particular release. Thus no-one else can check out or lock that specific part. If you are using *concurrent development* for the release, TeamConnection will not lock the part, and someone else can check out the same version of the same part from the release.

We highly recommend that you check out parts from the **TeamConnection - Parts** window. To check out a part use either the **Selected > Check out...** menu item or, better yet, the *pop-up* menu. In our case we used the *pop-up* menu and, after selecting the **Check out...** item, we got to the **Check Out Parts** window (see Figure 68).

Use the **Check Out Parts** window to check out parts from the server. When you check out parts, you must associate the parts with a work area, defect, or feature. You can check parts in and out of your work area without affecting the parts that are committed in the release. When you check in a part that you have checked out, the part is copied from your workstation to your work area on the server and changed on your workstation so that its part attribute is read-only. When you check out the part, two things happen:

- ☐ The read-only part on your workstation is changed so that the \$ character appears in the part's extension.
- ☐ The part you have checked out is on your workstation.

If you receive a message saying that you do not have the necessary authority, contact your family administrator and request PartCheckOut authority.

Note: As a rule, parts must be checked out before you can check them in. *Output parts* are an exception to this rule. *Output parts* cannot be checked out. A superuser can check in *output parts* that have not been previously checked out and thereby replace them.

The following fields and push buttons are available in the **Check Out Parts** window (see Figure 68):

Path names

Type the path names of the parts you want to check out. The path names can include both directory names and base names. If you selected path names from the **TeamConnection - Parts** window, those names appear in the field. You can change the information in this field before you press Enter.

Release

Type the name of the release to which the parts belong. You can import a release from the **Releases** window.

Work area

Type the name of the work area to which you want to assign the parts.

Destination directory

Type the directory path where you want to place the parts that you are checking out. If you specified a directory path in the Directory field of your Settings Notebook, the part will be checked out to this directory unless you specify a path in this field. You can change the directory path if it is not correct.

Break common link to parts locked elsewhere

Select this check box to break the common link in the release with which you are working. To break a part's link to more than one release, the part must be currently checked out or locked in another release. To break the common link when the part is not locked, use the check in action. To break the common link you must have authority beyond *PartCheckOut* authority; you must also be the component owner or have *PartForceOut* authority, which is defined in the component associated with the part.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

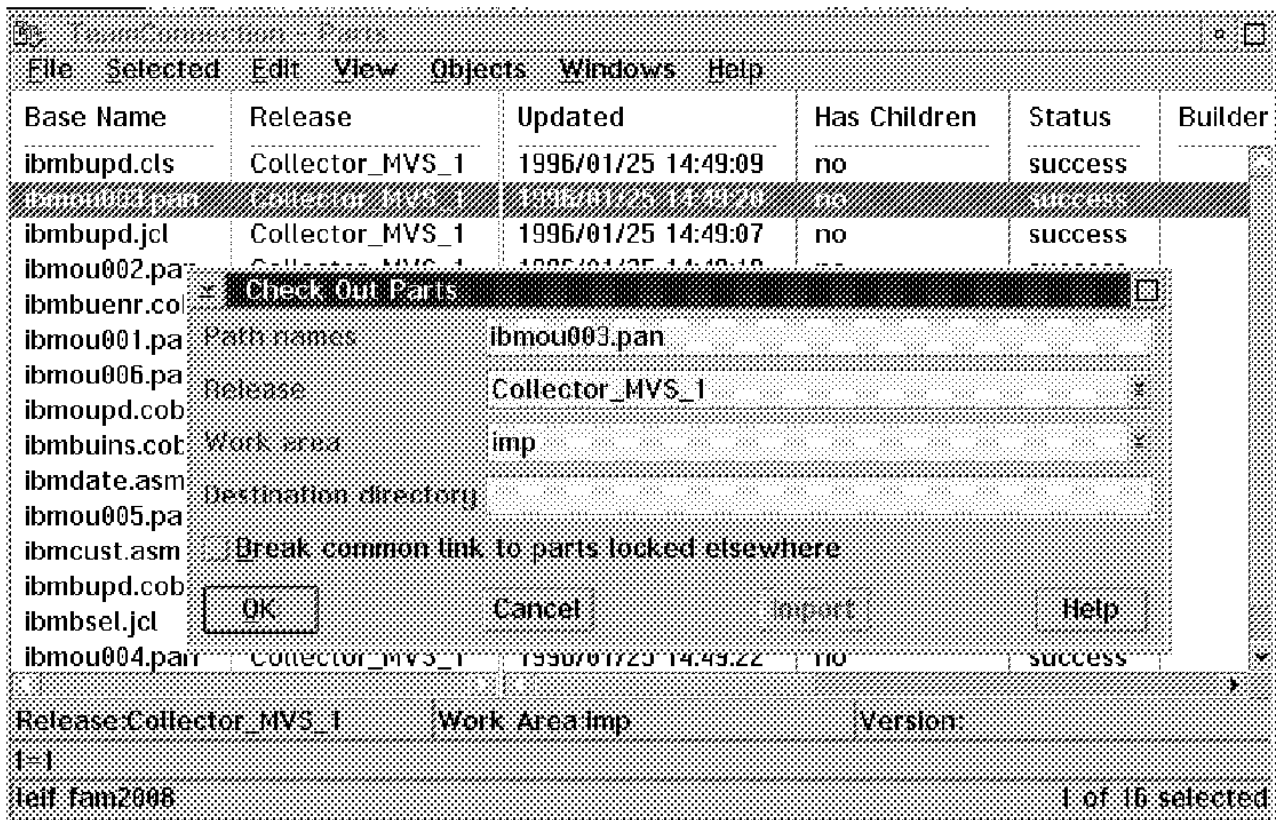


Figure 68. Check Out Parts Window

Command Line Interface Syntax

The following is the command line syntax for **checking out** parts:

```
teamc part -checkout Name ...
    -workarea Name -release Name -family Name
    [-force] [-stdout]
    [-relative Name | -top Name]
    [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-checkout <i>Name</i>	Retrieves a working copy of a specified part and locks it for editing purposes. Only the most recent version of a part can be checked out.
-workarea <i>Name</i>	Specifies the work area associated with a part (environment variable: TC_WORKAREA)

Attribute Flag and Argument	Description
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-force	Forces a break between common parts when using -lock , -checkout , -checkin , -delete , -recreate , -rename , or -undo .
-stdout	Redirects the specified part to standard output when extracting it from the TeamConnection server
-relative <i>Name</i>	Creates, checks in, checks out, or extracts the specified part relative to the directory location specified according to the complete path name of the part. Directories are created if necessary when extracting or checking out in order to copy the part by its full path name.
-top <i>Name</i>	Specifies the leading portion of the path name that is a subset of the current working directory on the client machine (environment variable: TC_TOP)
-become <i>Name</i>	Identifies the user ID you want to issue TeamConnection commands from, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command.

In our case, the command syntax would look like this:

```
teamc Part -checkout ibmou003.pan -release Collector_MVS_1 -workarea mp
```

We did not specify the **-relative** parameter, so TeamConnection will check out the part to whatever is specified in the **Relative directory** field on the **Environment** page in the **Settings** notebook (see Figure 34 in topic 6.1.1). As we did not specify anything in this field, TeamConnection will check out the part to the directory specified in the **Working directory** field on the **Environment** page in the **Settings** notebook. In our case, the part was checked out to the `D:\TEAMC\BIN` directory on the client machine.

If everything goes alright, you will be able to see in the **TeamConnection - Parts** window whether the specific part has been locked and by whom (see Figure 69); that is, you can see this in the **TeamConnection - Parts** window only if you are accessing the **TeamConnection - Parts** window from the same work area from which the checked out part was checked out. As you can see in Figure 69, even if you access the **TeamConnection - Parts** window from the same work area, but as a different user, you can still see whether the part has been checked out and by whom. In this case the work area is accessed by *felix*, and he can see that part *ibmou003.pan* has been checked out by user *leif*.

TeamConnection - Parts						
File	Selected	Edit	View	Objects	Windows	Help
Base Name	Release		Deleted	Type	Locked by	Mode
ibmbupd.cls	Collector_MVS_1	25 14:49:09		text		0600
ibmou003.pan	Collector_MVS_1	25 14:49:22		text	leif	0600
ibmbupd.jcl	Collector_MVS_1	25 14:49:07		text		0600
ibmou002.pan	Collector_MVS_1	25 14:49:18		text		0600
ibmbuenr.cob	Collector_MVS_1	25 14:44:47		text		0600
ibmou001.pan	Collector_MVS_1	25 14:49:16		text		0600
ibmou006.pan	Collector_MVS_1	25 14:49:26		text		0600
ibmoupd.cob	Collector_MVS_1	25 14:49:00		text		0600
ibmbuins.cob	Collector_MVS_1	25 14:48:47		text		0600
ibmdate.asm	Collector_MVS_1	25 14:49:13		text		0600
ibmou005.pan	Collector_MVS_1	25 14:49:24		text		0600
ibmcust.asm	Collector_MVS_1	25 14:49:11		text		0600
ibmbupd.cob	Collector_MVS_1	25 14:48:56		text		0600
ibmbasel.jcl	Collector_MVS_1	25 14:49:04		text		0600
ibmou004.pan	Collector_MVS_1	25 14:49:22		text		0600
Release: Collector_MVS_1 Work Area: mp Version: 1-1						
leif Jan2008						1 of 18 selected

Figure 69. Parts Window for Same Work Area but Different User

If you access the **TeamConnection - Parts** window from another work area (see Figure 70), you will not be able to see whether the part has been checked out or not.

TeamConnection - Parts						
File	Selected	Edit	View	Objects	Windows	Help
Base Name	Release		Deleted	Type	Locked by	Mode
ibmbupd.cob	Collector_MVS_1	25 14:48:56		text		0600
ibmbupd.jcl	Collector_MVS_1	25 14:49:07		text		0600
ibmcust.asm	Collector_MVS_1	25 14:49:11		text		0600
ibmdate.asm	Collector_MVS_1	25 14:49:13		text		0600

Figure 70. Parts Window for Different Work Area and User

To see whether a part has been checked out or not, you have to use the **TeamConnection - PartFull** window (see Figure 71). In this window you can see who has checked out which parts, from which work area, and from which version of that work area.

TeamConnection - PartFull						
File Selected Edit View Objects Windows Help						
Base Name	Release	Work Area	Current	Committed	Locked by	M
ibmou001.pan	Collector_MVS_1	imp	imp			
ibmou001.pan	Collector_MVS_1	testwa	testwa:2			
ibmou002.pan	Collector_MVS_1	imp	imp			
ibmou002.pan	Collector_MVS_1	testwa	testwa:2			
ibmou003.pan	Collector_MVS_1	imp	imp:2		test	
ibmou003.pan	Collector_MVS_1	testwa	testwa:2			
ibmou004.pan	Collector_MVS_1	imp	imp			
ibmou004.pan	Collector_MVS_1	testwa	testwa:2			
ibmou005.pan	Collector_MVS_1	imp	imp			
baseName like '*' and releaseName in ('Collector_MVS_1') order by baseName						
felix fam2008					1 of 32 selected	

Figure 71. TeamConnection - PartFull Window

You can also run the **Report** command to check which parts are locked, where they are locked, and by whom.

Command Line Interface Syntax

The **Report** command uses so-called views to retrieve information about components, releases, work areas, and parts. To get the information about which part is checked out and from where, use either the *PartFullView* or the *PartsOutView*. The following is the command line syntax for the **Report** command using views:

```
teamc report -view Name -family Name [-where Text] [-become Name]
          [-stanza | -raw | -table* | -long] [-verbose]
```

* Default

Attribute Flag and Argument	Description
-view <i>Name</i>	Specifies the database table or view you want to query. You can use a unique prefix abbreviation of the table and view names. The list of names you can view follows this table.

Attribute	Description
-----------	-------------

Flag and Argument	
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-where <i>Text</i>	Defines the selection criteria to query the specified table or view by using valid syntax
-become <i>Name</i>	Identifies the user ID you want to issue TeamConnection commands from, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-stanza	<p>Produces report output in stanza format:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Each database record is a stanza. <input type="checkbox"/> Each stanza line consists of a field and its corresponding values.
-table	<p>Produces report output in the following table format:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Each field is displayed as a column heading. <input type="checkbox"/> Field values appear under respective column headings. <input type="checkbox"/> Each row corresponds to one database record. <p>This is the default format of report output.</p>
-long	<p>Produces report output in stanza format, with additional important information shown in table format:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Each database record is a stanza. <input type="checkbox"/> Each stanza line consists of a field and its corresponding values.
-verbose	Specifies that you want to receive a confirmation message after you issue this command

TeamConnection at Large
Check Out a Part



6.7.2 Edit a Part

Editing a part is very much like checking out a part. In fact the only difference is that when you edit a part, you not only check out the part, but you also invoke an editor of your choice for that particular part. The so-called editor can be a small command file (such as a REXX command file) that enables you to edit any *textual* or *binary* file, provided you have a tool that can handle the particular part you want to edit.

Figure 72 shows a REXX command file that enables us, using the same editor, to "edit" such files as Freelance Graphics files, BookManager Read/2 files, and ordinary text files. This REXX command file actually chooses between two text editors, depending on the extension of the part that is being edited. If we have a part with the extension of *.CPP*, we will invoke another REXX command file, *lxsync.cmd*. This command file in turn invokes the LPEX editor, which is a live parsing editor that uses colors. For all other part extensions (except *.PRS* and *.BOO*), we invoke our favorite editor, the EPM editor.

```

/*****
/*  medit.cmd - Invokes appropriate "editor" according to
/*  file extension.
/*  Author:  Leif Trulsson - ITSO San Jose    951011
/*
*****/

  trace 'o'

  arg file_name

  env='OS2ENVIRONMENT'

  w_name = toupper(file_name)

  select
    /* If we have a .PRS file, invoke Freelance */
    when pos('.PRS',w_name) > 0 then do
      /* Used to find Freelance, set in CONFIG.SYS*/
      flg_path=getenv('FLGPATH')
      if flg_path = '' then
        flg_path = 'D:\FLG'
      flg_path||'\flg presentation' file_name
    end /* Do */
    /* If we have a .CPP file, invoke LPEX */
    when pos('.CPP',w_name) > 0 then
      'lxsync' file_name
    /* If we have a .BOO file, invoke BookManager*/
    when pos('.BOO',w_name) > 0 then
      'epaibm0' file_name
    otherwise
      'epm' file_name /* Otherwise use the EPM editor */
  end /* select */

  exit(0)

toupper:
  parse upper arg cmd /* Convert to upper case */
  return cmd

getenv:
  /* Get environment variable*/
  arg envvar
  return value(envvar,, 'OS2ENVIRONMENT')

```

Figure 72. Medit REXX Command File

We highly recommend that you check out, edit, and check in parts from the **TeamConnection - Parts** window. By doing so, you always know from which work area you checked out or edited a part.

Use the **Edit Part** window (see Figure 73) to edit a part that is on either your workstation or the server. If you edit a part on the server, check out the part. To use your favorite editor, enter the appropriate command in the Edit command field. The following fields and push buttons are available in the **Edit Part** window:

Path name

Type the path name of the part you want to edit. The path name can include both the directory name and base name.

Release

Type the name of the release that is associated with the part.

Work area

Type the name of the work area associated with the part you want to edit.

Break common links to parts locked elsewhere

Select this check box to break the common link in the release with which you are working.

Edit command

Type the command you want to use to invoke your editor from within TeamConnection. For example, if you invoke your editor by typing *Edit*, type *Edit* in this field. You can also specify this command on the **Setup** page of the **Settings** notebook.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

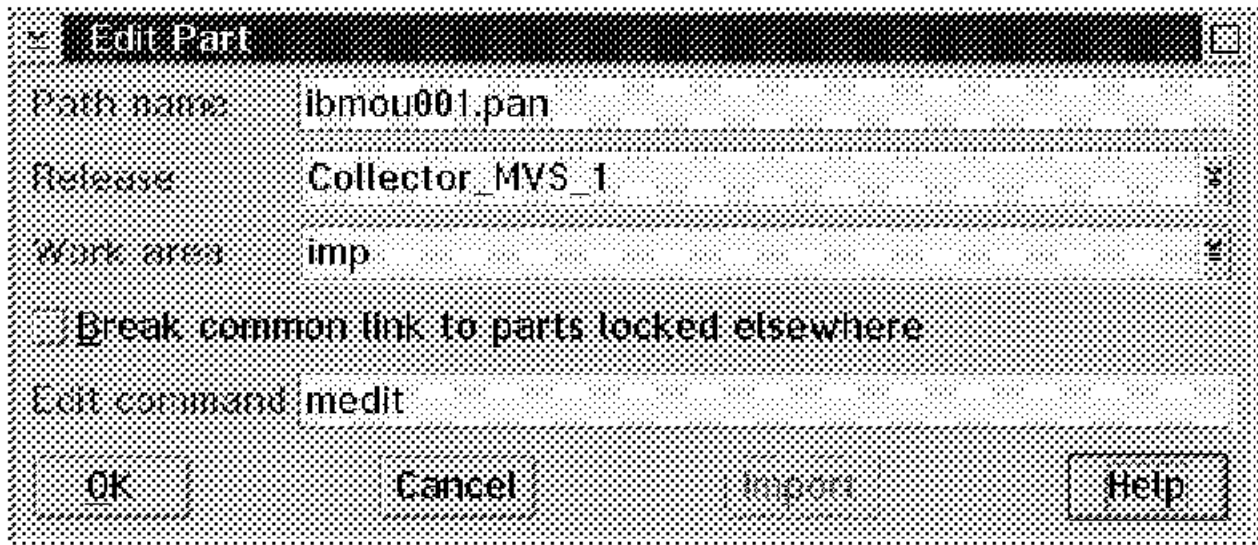


Figure 73. Edit Part Window

Use the **Finished Editing Part** window (see Figure 74) to specify what you want done with the file that you just finished editing. From this window, you can either choose to do nothing with your changes, unlock the TeamConnection part, or check the part into TeamConnection.

The following fields and push buttons are available in the **Finished Editing Part** window:

Check it in

Select this radio button to indicate that you want to check in the part.

Unlock it

Select this radio button to indicate that you want to unlock the part without checking in your changes.

Path name

Type the path name for the part that you just finished editing. The path name can include both the directory name and base name.

Release

Type the name of the release that is associated with the part.

Work areas

Type the name of the work areas for the part. The information you type determines which work area is associated with the view.

Directory

Type the source directory of the file you just edited.

Note: The default for this field is the *TMP* directory defined in your *CONFIG.SYS* file. This is to avoid corrupting any files in the working directory that might be using the same name.

Check in

Specify the following information if you want to check in the part:

Common releases

Type the names of any common releases. If the parts are common to multiple releases and you want them to remain common, specify any other releases for which you want the parts to be checked in. You can import selected releases from the **Releases** window.

Remarks

Type comments directly into this field.

Edit

You can select the **Edit** push button to type lengthy remarks or to insert text from a file.

Retain lock

Select this check box to specify that you want to perform all of the check-in actions but do not want to unlock the parts and allow other users to check them out.

Break common link

Select this check box to break the common link in all releases in which the parts are common except for releases specified in the Release and Common releases fields. If you break the common link, the part becomes a shared part among the releases in which the link was broken. Breaking the common link requires authority beyond *PartCheckIn* authority; you must be the component owner or have *PartForceIn* authority, which is defined in the component associated with the part.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

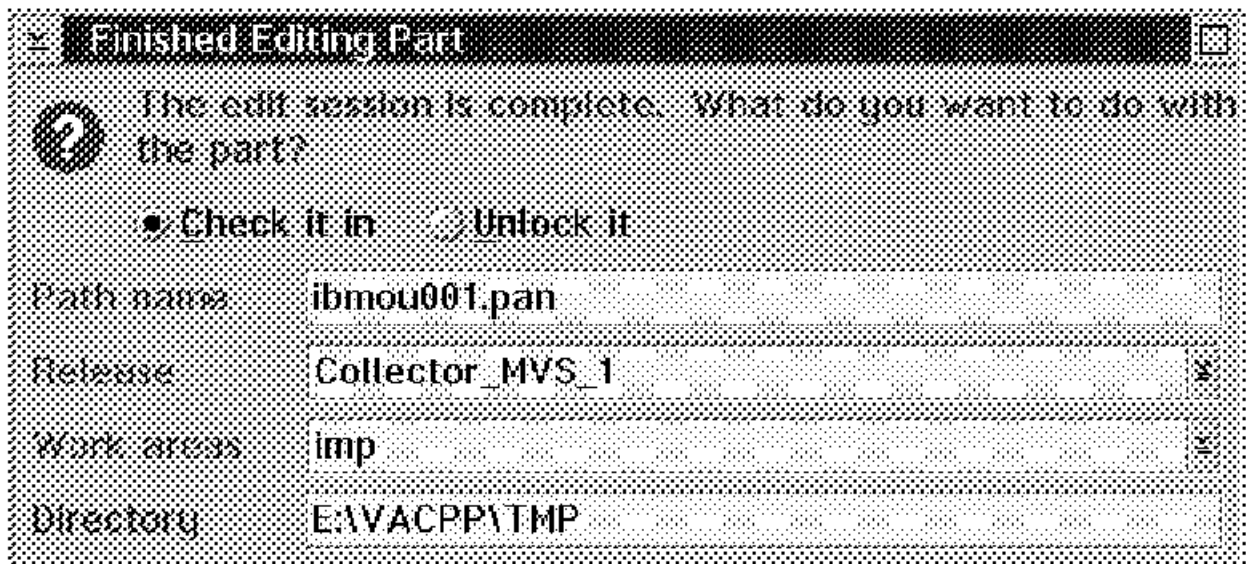
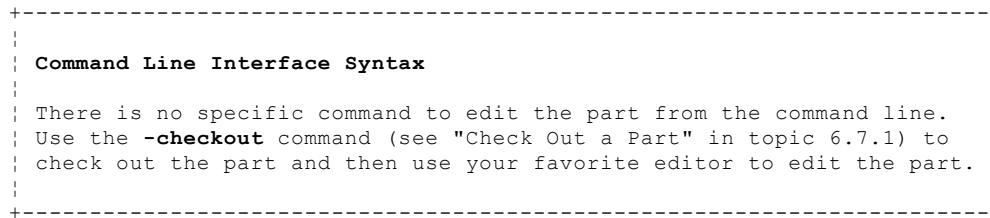


Figure 74. Finished Editing Part Window



6.7.3 Check in a Part

When you are done with the changes to the part that you checked out, you have to check it in again. The best way is to select the **Check in...** menu item from the pop-up menu in the **TeamConnection - Parts** window (see Figure 75). You do this by selecting the part using the right mouse button.

When you access the pop-up choices for a part, you notice that the list varies depending on the part you select. TeamConnection lists only those options that are valid based on the part you selected and its current state.

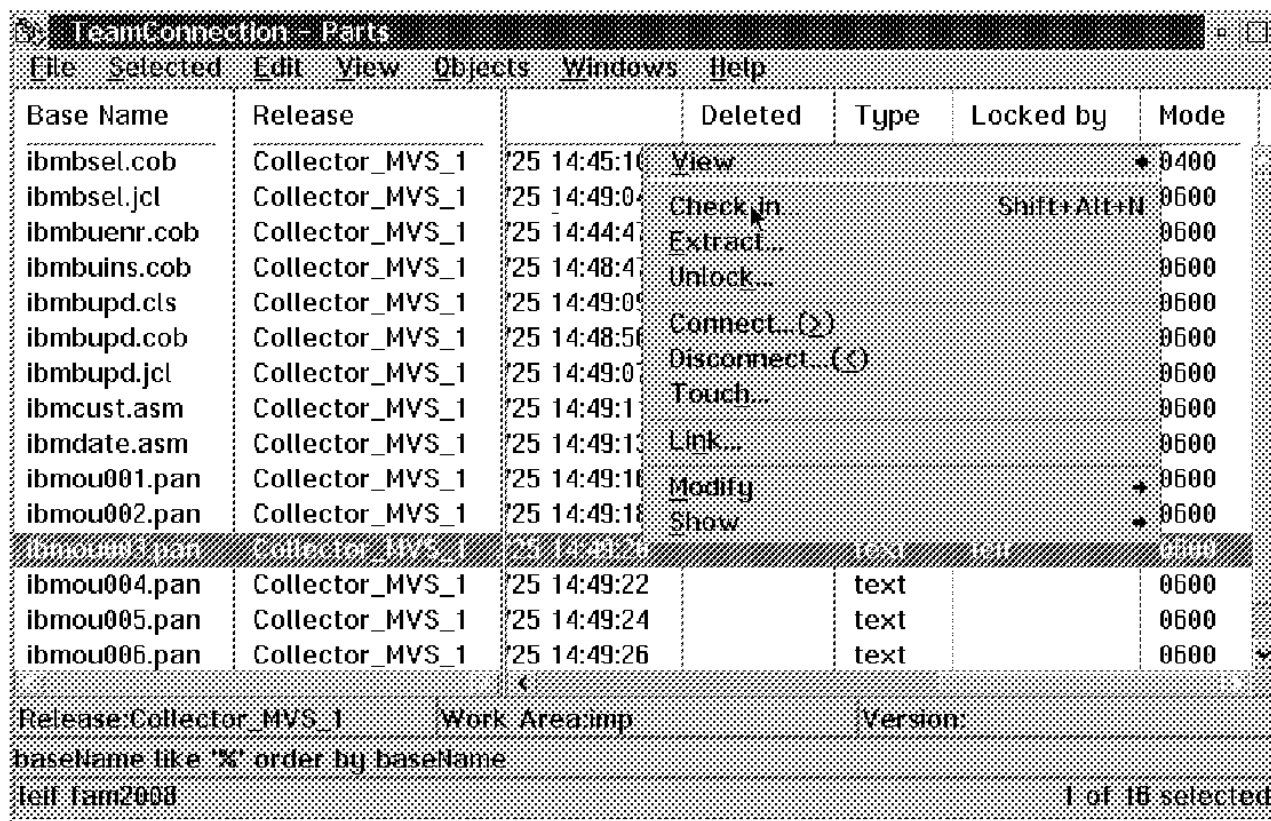


Figure 75. Pop Up Choices for Checked-Out Part

Use the **Check In Parts** window (see Figure 76) to check in parts from:

- ☐ Your work area
- ☐ Somebody else's work area (provided you have the authority)
- ☐ A defect or feature

You must have a work area to check in or create parts. When you check in a part that you have checked out, the part is:

- ☐ Copied from your workstation to your work area on the server
- ☐ Changed on your workstation so that its part attribute is read-only
- ☐ Changed on the server so that it is unlocked and other users can check out the part (only if the part is checked out from the same work area to which it was checked in).

If you receive a message saying that you do not have the necessary authority, contact your family administrator and request *PartCheckIn* authority.

Note: As a rule, parts must be checked out before you can check them in. *Output parts* are the exception to this rule. *Output parts* cannot be checked out. A superuser can check in *output parts* that have not been previously checked out.

The following fields and push buttons are available in the **Check In Parts** window:

Path names

Type the path names of the parts you want to check in. The path names can include both directory names and base names. If you selected path names from the **Parts** window, those names appear in the field. You can change the information in this field before

you press Enter.

Release

Type the name of the release to which the parts belong. You can import a selected release from the **Releases** window.

Work areas

Type the name of the work area from which you want to check out the parts.

Source directory

Type the directory path where the parts that you are checking in are located. If you specified a directory path in the *Directory* field of the **Settings** notebook, that path appears in this field. You can change the directory path if it is not correct. When you append a part name to the source directory, that specific part is checked in.

Common releases

Type the names of any common releases. If the parts are common to multiple releases and you want them to remain common, specify any other releases for which you want the parts to be checked in. You can import selected releases from the **Releases** window.

Remarks

Type comments directly into this field. These remarks are appended to other remarks to create a history trail of changes to the part.

Edit

You can also select the **Edit** push button to type lengthy remarks or to insert text from a file.

File type

Type the file type of the part you are checking in. Select one of the following:

Text

Specifies a part with ASCII- or EBCDIC-encoded data

Binary

Specifies a part other than ASCII- or EBCDIC-encoded data. Examples of such parts are graphical, object code, or executable parts. To create a placeholder part, select **Binary**.

None

Specifies that a part is not associated with this object

No change

Specifies that the file type does not change

Source

Type the location from which you are getting the contents of the part you are creating in TeamConnection. You can select one of the following:

Same

Specifies that the contents of the part are from an identically named part on your workstation

Copy from

Specifies that the contents of the part are from a part on your workstation with a different name. You must type the name of this part in the *Source part* field.

Source file

Type the name of the file whose contents you want to use when creating a new part.

Break common link

Select this check box to break the common link in all releases in which the parts are common except for releases specified in the Release and Common releases fields. If you break the common link, the part becomes a shared part among the releases in which the link was broken. To break the common link requires authority beyond *PartCheckIn* authority, you must be the component owner or have *PartForceIn* authority, which is defined in the component associated with the part.

Retain lock

Select this check box to specify that you want to perform all of the check in actions except you do not want to unlock the parts and allow other users to check them out.

Select

Displays the **Select Source File** window. From this window, you can select a part name from your workstation. The part you select appears in the Source part field.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

Check In Parts

Path names:

Release:

Work areas:

Source directory:

Common releases:

Remarks:

Edit...

File type: ☐ Text ☐ Binary ☐ None ☒ No change

Source: ☒ Same ☐ Copy from

Select

☐ Break common link ☐ Retain lock

OK Cancel Import Help

Figure 76. Check In Parts Window

When trying to check in the *ibmou003.pan* part, we got an error (see Figure 77), that is, we did not succeed in checking in the part. We got the message shown in Figure 77 because we had previously created another work area, *testwa*, and used the *imp* work area as the source (not the release). Therefore these two work areas are actually handling the same versions of the parts. In other words, there is a commonality between the two work areas. The two work areas are what we call **common**.



Figure 77. Check In Parts Error Window

The error that we got tells us to take an action (make a selection). We can either maintain commonality or break it. In our case, we chose to break the commonality. Therefore the update that we made to the *ibmou003.pan* part will not be seen in work area *testwa*. We tell TeamConnection to break the commonality by selecting the **Break common link** check box in the **Check In Parts** window (see Figure 78) and we select the **OK** push button to check in the part.

Figure 78. Selecting Break Common Link in the Check In Parts Window

Command Line Interface Syntax

The command line interface syntax for checking in a part looks like this:

```
teamc part -checkin Name ... -release Name -family Name
        -workarea Name ...
        [-common Name ...] [-force] [-retainLock] [-remarks Text]
        [-binary | -text | -none] [-stdin | -from fileSpec]**
        [-relative Name | -top Name]
        [-become Name] [-verbose]
```

** Only valid with file type of text or binary.

Attribute Flag and Argument	Description
-----------------------------------	-------------

TeamConnection at Large Check in a Part

-checkin <i>Name</i>	Submits the changes made to a specified part to the TeamConnection server and unlocks the part. (Any associated work areas must be in the <i>fix</i> state, and the associated fix records in the <i>ready</i> or <i>active</i> state.)
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-workarea <i>Name</i>	Specifies the work area associated with a part (environment variable: TC_WORKAREA)
-common <i>Name</i>	Specifies the releases in which common parts are to be maintained or whether the specific part change is to apply to all releases in which the part is common. All releases must be specified unless the -force flag is specified as well.
-force	Forces a break between common parts when using -lock , -checkout , -checkin , -delete , -recreate , -rename , or -undo
-binary	Indicates that the part being created is a binary file. The default file type is <i>text</i> .
-text	Indicates that the part is being created with ASCII- or EBCDIC- encoded data

Attribute Flag and Argument	Description
-none	Indicates that the part being created will never contain any data; for example, the part might be a <i>collector</i> object

TeamConnection at Large Check in a Part

-stdin	Redirects from standard input
-from <i>fileSpec</i>	Indicates the location of file contents
-relative <i>Name</i>	Creates, checks in, checks out, or extracts the specified part relative to the directory location specified according to the complete path name of the part. Directories are created if necessary when extracting or checking out in order to copy the part by its full path name.
-top <i>Name</i>	Specifies the leading portion of the path name that is a subset of the current working directory on the client machine (environment variable: TC_TOP)
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

Using the above syntax, our check-in would then look like this:

```
teamc Part -checkin ibmou003.pan -release Collector_MVS_1
          -workarea imp -force -from D:\TEAMC\BIN\ibmou003.pan
```

6.7.4 Lock a Part

When we checked out a part (see "Check Out a Part" in topic 6.7.1), we noticed that the part was locked. TeamConnection also has a way to **lock** a part without checking it out. Use the **Lock Parts** window (see Figure 79) to prevent other users from checking out the part. You can only lock the current version of a part. This is useful when you want to limit access to parts in the library, but you do not necessarily want to check out the parts.

The following fields and push buttons are available in the **Lock Parts** window:

Path names

Type the names of the parts you want to lock. The path names can include both directory names and base names.

Type

Type the part type. For example *File* is a part type. Parts with a part type of *File* can exist on the workstation. If you do not specify a part type, this field defaults to *File*. Depending on your development environment, you may have parts that have a part type other than *File*.

Release

Type the name of the release to which the parts that you want to lock belong. You can import a release from the **Releases** window.

Work area

Type the name of the work area for the parts you want to lock.

Break common link to parts locked elsewhere

Select this check box if you want multiple locks on parts that are common in other releases. The default is not to perform multiple locks on common parts. To perform this action, you must be the component owner or have *PartLockForce* authority defined in the component for the part.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

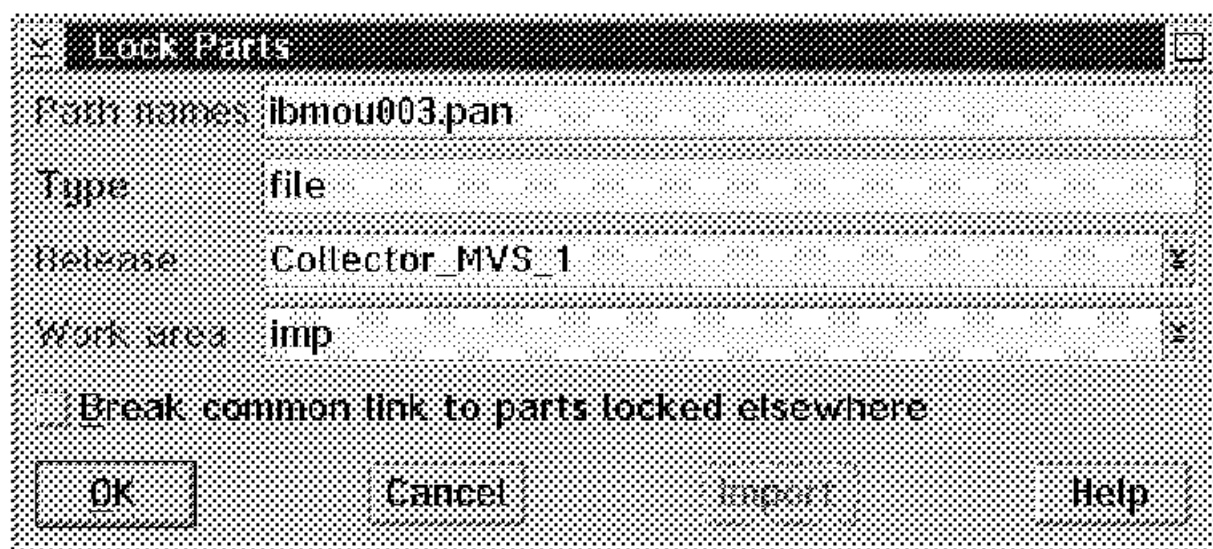


Figure 79. Lock Parts Window

Command Line Interface Syntax

The command line interface syntax for locking a part looks like this:

```
teamc part -lock Name ...
  -workarea Name
  -release Name -family Name [-force]
  [-type Name]
  [-top Name] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-lock <i>Name</i>	Locks a part in the TeamConnection server. This prevents other users from checking out the part. Only the current version of a part can be locked.
-workarea <i>Name</i>	Specifies the work area associated with a part (environment variable: TC_WORKAREA)
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-force	Forces a break between common parts when using -lock , -checkout , -checkin , -delete , -recreate , -rename , or -undo .
-type <i>Name</i>	Specifies the type of the part; for example, all parts created through the TeamConnection command line have the type of <i>File</i> .
-top <i>Name</i>	Specifies the leading portion of the path name that is a subset of the current working directory on the client machine (environment variable: TC_TOP)
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).

Attribute Flag and Argument	Description
-verbose	Specifies that you want to receive a confirmation message after you issue this command

Using the above syntax, our lock would then look like this:

```
teamc Part -lock ibmou003.pan -type file -release Collector_MVS_1
-workarea imp
```

When a part has been checked out, edited, or locked, it will stay locked in the release until you either **integrate** the work area from where the part was checked out, edited, or locked or unlock the part specifically. Then, if somebody else tries to check out, edit, or lock the same part from another work area, he or she will get a message like that shown in Figure 80.

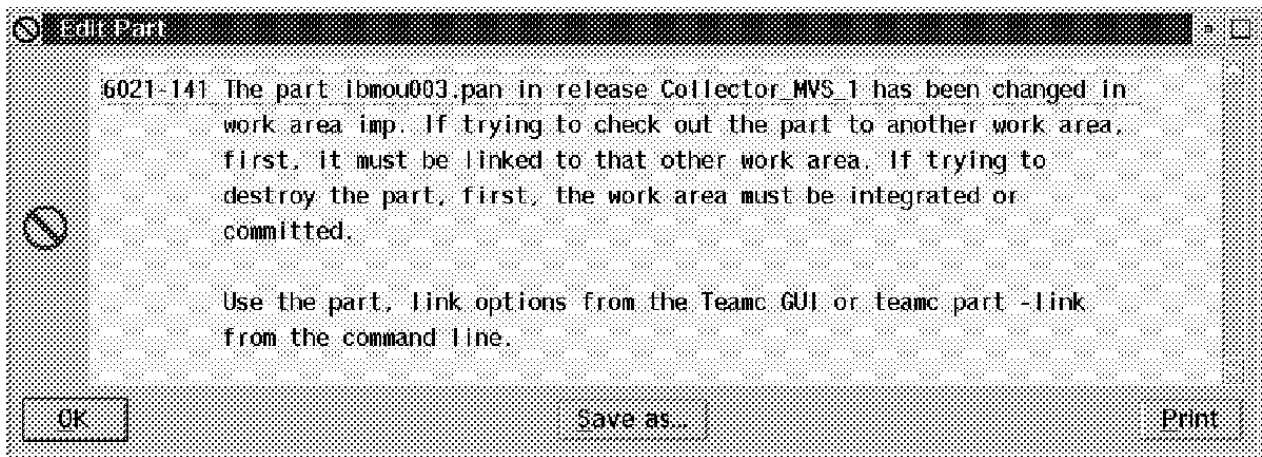


Figure 80. Edit Part Information Window

6.7.5 Unlock a Part

Use the **Unlock Parts** window (see Figure 81) to unlock a part that is checked out (or that has been previously locked) so that other users can check out the part. You might unlock a part for any of the following situations:

- ☐ You check out a part but decide you do not want to change it. Unlocking the part prevents having to check in the part.
- ☐ Someone else has checked out a part and then leaves for the day. You decide that you have to edit the part, and you have the authority to unlock the part. If you unlock the part and make your changes, you have to reconcile any part changes you make with the changes made by the user who initially checked out the part.

If you receive a message saying that you do not have the necessary authority, contact your family administrator and request *PartUnlock* authority.

The following fields and push buttons are available in the **Unlock Parts** window:

Path names

Type the names of the parts you want to unlock. The path names can include both directory names and base names.

Type

Type the part type. For example, *File* is a part type. Parts with a part type of *File* can exist on the workstation. If you do not specify a part type, this field defaults to *File*. Depending on your development environment, you may have parts that have a part type other than *File*.

Release

Type the name of the release for the parts that you want to unlock. You can import a selected release from the **Releases** window.

Work area

Type the name of the work area for the parts you want to unlock.

Source directory

Type the directory on your workstation on which the parts are located. This field is initialized with the value specified through the **Environment** page of the **Settings** notebook.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

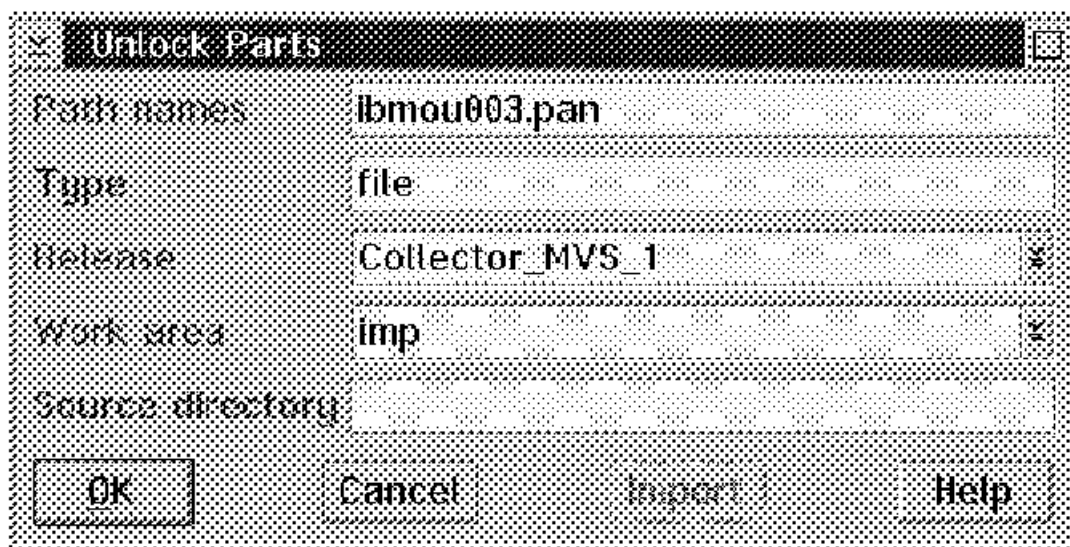


Figure 81. Unlock Parts Window

Command Line Interface Syntax

The command line interface syntax for unlocking a part looks like this:

```
teamc part -unlock Name ...  
-workarea Name  
-release Name -family Name  
[-relative Name | -top Name]  
[-type Name]  
[-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-unlock <i>Name</i>	Unlocks a part that is checked out so that it is no longer reserved for editing purposes, or unlocks a part that has been previously locked by using -lock
-workarea <i>Name</i>	Specifies the workarea associated with a part (environment variable: TC_WORKAREA)
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-force	Forces a break between common parts when using -lock , -checkout , -checkin , -delete , -recreate , -rename , or -undo
-relative <i>Name</i>	Creates, checks in, checks out, or extracts the specified part relative to the directory location specified according to the complete path name of the part. Directories are created if necessary when extracting or checking out in order to copy the part by its full path name.

Attribute Flag and Argument	Description
-top <i>Name</i>	Specifies the leading portion of the path name that is a subset of the current working directory on the client machine (environment variable: TC_TOP)
-type <i>Name</i>	Specifies the type of the part; for example, all parts created through the TeamConnection command line have the type of <i>File</i>
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

Using the above syntax, our unlock would then look like this:

```
teamc Part -unlock ibmou003.pan -type file -release Collector_MVS_1
-workarea imp
```

6.7.6 Extract a Part

Use the **Extract Parts** window (see Figure 82) to copy TeamConnection parts to a disk. Extracting a part does not lock the part in the database; therefore, you can extract a copy of a part that is currently checked out to another user. You must be careful when using extracted parts. Do not extract a part that you want to change. If you have to make changes to a part, you must check out the part from the database so that it is locked. Otherwise, changes could be lost. If you receive a message saying that you do not have the necessary authority, contact your family administrator and request *PartExtract* authority.

The following fields and push buttons are available in the **Extract Parts** window:

Path names

Type the names of the parts you want to extract. The path names can include both directory names and base names. If you selected Path names from the **Parts** window, those names appear in the field. You can change the information in this field before you press Enter.

Release

Type the name of the release to which the parts belong. You can import a selected release from the **Releases** window.

Work area

Select the **Work area** push button and type the name of the work area to which you want to assign the parts. If you do not specify a work area or version, the latest integrated version is extracted.

Version

Select this push button to associate the part with a particular version. You can specify a version that is different from the current version. If you do not specify a work area or version, the latest integrated version is extracted.

Destination directory

Type the directory path where you want to put the parts that you are extracting. If you specified a directory path in the *Directory* field of your **Settings** notebook, that path appears in this field. You can change the directory path if it is not correct.

Read-only

Select this check box so that the extracted parts are read-only.

Expand keywords

Select this check box to prevent substitution of assigned values in place of keywords. The default is to substitute values for the keywords. Keywords are used with text parts, not with binary parts. For more information about supported keywords, refer to the *IBM TeamConnection for OS/2 User's Guide*, SC34-4499.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

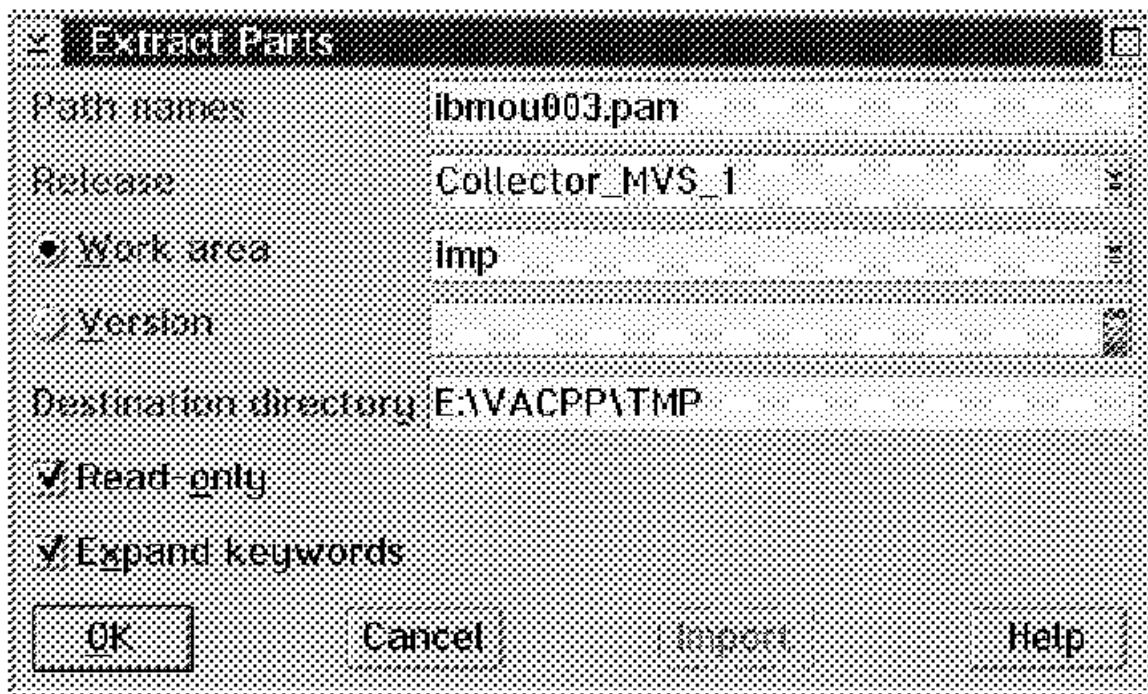


Figure 82. Extract Parts Window

Command Line Interface Syntax

The following is the command line interface syntax for extracting a part:

```
teamc part -extract Name ...
      -release Name -family Name
      [-workarea Name*** | -version Name]
      [-nokeys] [-relative Name | -top Name] [-stdout]
      [-dmask Octal_number] [-fmask Octal_number]
      [-become Name] [-verbose]
```

*** Required if part has not been committed.

Attribute Flag and Argument	Description
-extract <i>Name</i>	Retrieves a copy of a specified part. The current version is extracted by default.
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-workarea <i>Name</i>	Specifies the work area associated with a part (environment variable: TC_WORKAREA)

-version <i>Name</i>	Specifies the version of the part you want to extract, view, or link
-nokeys	Indicates that keywords should not be expanded when a part is extracted
-relative <i>Name</i>	Creates, checks in, checks out, or extracts the specified part relative to the directory location specified according to the complete path name of the part. Directories are created if necessary when extracting or checking out in order to copy the part by its full path name.
-top <i>Name</i>	Specifies the leading portion of the path name that is a subset of the current working directory on the client machine (environment variable: TC_TOP)

Attribute Flag and Argument	Description
-stdout	Redirects the specified part to standard output when extracting it from the TeamConnection server
-dmask <i>Octal_Number</i>	Specifies the read, write, and execute directory permissions in octal notation for the extracted part Note: Although the OS/2 client accepts -dmask , the flag has no effect.
-fmask <i>Octal_Number</i>	Specifies the read and write file permissions in octal notation for the extracted parts. A value of 700 turns off the read-only attribute. A value of 00 extracts a part according to the <i>fmode</i> value. The -fmask flag overrides the -fmode setting.

TeamConnection at Large

Extract a Part

-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
----------------------------	--

-verbose	Specifies that you want to receive a confirmation message after you issue this command
-----------------	--

Using the above syntax, our extract would then look like this:

```
teamc Part -extract ibmou003.pan -release Collector_MVS_1
           -workarea imp -relative E:\VACPP\TMP
```

6.8 Creating Versions

Versioning is an insurance policy. In TeamConnection you do not version the individual parts on an individual basis; you version the whole work area. TeamConnection does not version your parts every time you do a check-in or an unlock. Instead, TeamConnection versions work areas, drivers, and releases. You create new versions by:

- ☐ *Freezing* your work area
- ☐ *Integrating* the work area (making it the most current view of the release)
- ☐ *Adding* the work area as a *driver member* to a driver (making it the most current view of the driver)
- ☐ *Committing* a driver (making it the most current view of the release and integrating all driver members; see Chapter 9, "Using TeamConnection's Integrated Problem Tracking and Change Control" in topic 9.0)

By *freezing* the work area, you know that the parts currently visible in the work area will be retained in their current form, and you can then go back to this version, if you have to. Use the **Freeze Work Areas** window (see Figure 83) to take a current version of the parts in the work area and save them in the TeamConnection family (database). When you freeze a work area, TeamConnection creates a new version on the server and saves the version of the parts in the work area into this new version. Thus the work area changes to the TeamConnection family are saved. This version represents a "snapshot in time" of the parts in your work area.

The following fields and push buttons are available in the **Freeze Work Area** window:

Work areas

Type the names of the work areas you want to freeze.

Releases

Type the release with which the work area is associated.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

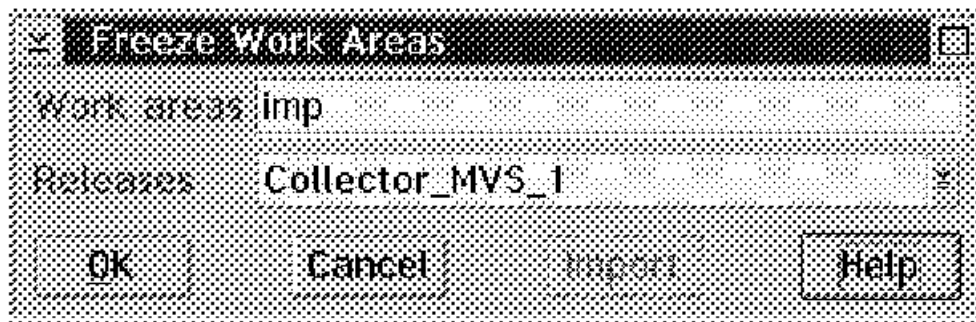


Figure 83. Freeze Work Areas Window

Changes associated with the work area are not visible to the release until you commit the work area. If your release does not have the *driver process* enabled (see Chapter 9, "Using TeamConnection's Integrated Problem Tracking and Change Control" in topic 9.0), TeamConnection commits the work area implicitly through the **workarea -integrate** command. Use the **Integrate Work Areas** window (see Figure 84) to integrate a work area into the release.

The following fields and push buttons are available in the **Integrate Work Areas** window:

Work areas

Type the names of the work areas you want to integrate.

Releases

Type the names of the releases for the work areas you are integrating.

Ignore build status

Select this check box to ignore the build status of the parts and force the integrate for the work areas.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

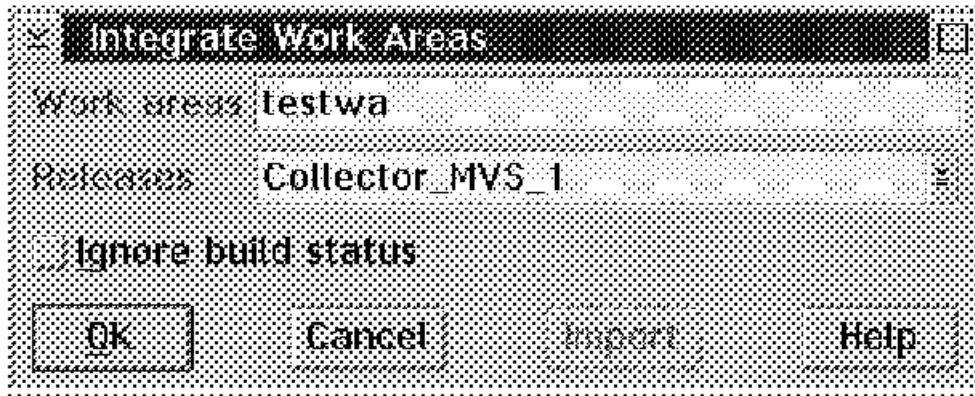


Figure 84. Integrate Work Areas Window

Command Line Interface Syntax

The following is the command line interface syntax for freezing and integrating a work area:

```
teamc workarea -freeze Name ... -release Name ...
                  -family Name [-become Name] [-verbose]

teamc workarea -integrate Name ... -release Name ...
                  -family Name [-force] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-freeze Name ...	Saves the state of a work area
-integrate Name ...	Changes the state of the specified work areas from fix to the next valid state governed by the release's process. For a release whose process includes the driver subprocess, this action is valid only if part changes were not made for the work area and the work area is not committed in a driver.

-release <i>Name ...</i>	The releases for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-force	Continues with the operation even if build outputs contained in the work area are out-of-date
-become <i>Name</i>	Identifies the user ID from which want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

Using the above syntax, our freeze would then look like this:

```
teamc WorkArea -freeze imp -release Collector_MVS_1
```


"Built to last."

-- Famous Slogan --

In "Integrated Build" in topic 2.4 we introduce you to TeamConnection's integrated build function. It is typically the task of your **build administrator** to create the **build scripts**, the **builders** and **parsers**, and the application **build tree** and configure the **build agents** and **build processors**. You would probably also have a dedicated person who does the *final* build of your application. But the individual *unit* builds will probably be done by each individual developer.

In this chapter we show you what it takes to build an application by using the integrated build function. We explain how to:

- ☐ Work with build scripts
- ☐ Organize build pools and build environments
- ☐ Organize your build agents and build processors
- ☐ Create a builder
- ☐ Create a parser
- ☐ Create a build tree
- ☐ Use a NULL builder
- ☐ Build an application

But first we take a look at the different build users and their roles.

Subtopics

- 7.1 Build Users and Their Roles
- 7.2 Working with Build Scripts
- 7.3 Organizing Build Pools and Build Environments
- 7.4 Organizing the Build Agents and Build Processors
- 7.5 Creating a Builder
- 7.6 Creating a Parser
- 7.7 Creating a Build Tree
- 7.8 Using a NULL Builder
- 7.9 Building an Application

7.1 Build Users and Their Roles

From a build point of view, we are looking at three different roles:

- ☐ The build administrator
- ☐ The developer
- ☐ The project administrator

These different roles have different tasks, and they are as follows:

The build administrator: Would cater for the following:

Parser

The parsers provided can be changed for the specific environment. It is also possible to define your own parser (for example, use the Make make file and load the dependencies through APIs into the database). The parsers are provided on an *as-is* basis.

Builder

Defining builders is a one-time job per type of translation. Decisions can be made to split multiple translator steps or combine them (for example, combine precompile, compile, and link steps together in one build script or separate them). The build scripts are provided on an *as-is* basis. A build step is not necessarily a "translate"; one might consider copying the resulting load module to a library (for example, on MVS) or invoking *Gather/2* and/or *NVBridge/2* functionality as build steps (on OS/2).

Topology

You can have different topologies for the build servers for distributed build (different platforms, for example, MVS and OS/2). Parallel builds are also possible. If there are lots of compiles to perform, the work can be split and performed on different machines. Builders can be bundled into **pools**, which are serviced by **agents** dispatching and controlling the work within a given *pool*. Obviously the project administrator has to decide on the overall *topology* of TeamConnection components.

The developer: Can be an application developer but can also be, for example, a document writer who writes the *.INF* files that make up the online documentation. Some of the tasks that the developer will handle are:

Dependencies

The developer is responsible for the definition of the dependencies apart from the ones that can be found with the parsers. Thus the developer has to specify the relationships between build objects that need translation (for example, relationship between executable and called application parts). This specification can be done through the GUI or API and is transparent for VisualGen developers. When defining a file, the developer must associate the parser and builder if applicable. When requesting a build, the target must be specified.

Build

Application developers usually perform builds at the development level (unit test).

The project administrator: A larger project will probably have a project administrator who will create the initial component structure for the project, grant access to appropriate users in the project, create notification lists, and be responsible for the *application build*.

Build

A project administrator usually does the application build for test, production, and maintenance.

7.2 Working with Build Scripts

A **build script** is essentially a binding between TeamConnection and a transform tool. The build script invokes the tool. Because of the way TeamConnection uses build scripts to invoke tools, it is highly extensible and can be used with a large variety of build tools, such as:

- ☐ Application generators (for example, VisualGen)
- ☐ 3GL compilers
- ☐ Linkers
- ☐ Object-oriented C++ compilers
- ☐ Preprocessors such as for DB2 and CICS
- ☐ Document processors

In OS/2, a build script is usually a command file. It can be as simple as a single string that invokes the tool. In MVS, it is usually a JCL fragment, but it can also be a REXX or CLIST file. The JCL job stream can do anything that a normal JCL job stream does. For example, a build script designed to move an application into test might copy the application executable into a test loadlib along with a set of test cases.

Subtopics

7.2.1 For OS/2

7.2.2 For MVS

7.2.3 For Parsers

7.2.1 For OS/2

Figure 85 shows a sample build script for a C++ compile on OS/2.

```

/*-----*/
/* PROGRAM:  FHBOCOMP.CMD                                */
/* IBM C Set++ Sample Build Script TeamConnection        */
/* ICC C++ Compile Only                                  */
/*                                                     */
/*-----*/
  parse arg parms

  environ  = 'OS2ENVIRONMENT'
  input    = VALUE('TC_INPUT',,environ)
  output   = VALUE('TC_OUTPUT',,environ)

  translator  = 'icc' /* specify the translator to be invoked */
  default_parms = '/C+' /* specify any default parms to be used */
                        /* for the compile                        */

  if parms = '' then parms = default_parms

  call invokeTranslator

exit result /* exit with the rc return by the translator */

/*-----*/
/* Routine: invokeTranslator                                */
/*                                                     */
/*-----*/
invokeTranslator:

  /* This build script assumes only one input and */
  /* one output are passed in.                    */

  translator parms '/Fo'||output input

return rc

```

Figure 85. Sample Build Script for C++ Compile on OS/2

You can specify parameters that will be passed to the build function in three ways.

As attributes of a builder

Builder parameters are passed to the build script after variable substitution is performed. Variables are substituted based on the following syntax:

`$(variable_name)`

To pass parameters to your build script, specify them in the Parameters attribute of the builder. TeamConnection sets these variables before invoking the build script. You can use the following TeamConnection environment variables:

TC_FAMILY

The TeamConnection family

TC_RELEASE

The release of the parts that are being built

TC_LOCATION

The current directory where the build script runs

TC_INPUT

A list of the TeamConnection parts that are input to the object being built

TC_INPUTTYPE

Identifies each input type. The default is file.

TC_OUTPUT

A list of the parts that are being built in this build event

TC_OUTPUTTYPE

Identifies each output type. The default is file.

TC_WORKAREA

The name of the work area in which the build is being performed.

You can also define other variables. These can be set when you start the build by specifying a value for the Parameters attribute in the *part -build* command (from the command line or through the GUI). These variables are passed to the build script. They are also used to set OS/2 environment variables before the build script is invoked.

As attributes of a part in the build tree

Parameters that are unique to a particular part are specified on the *part -create* and *part -modify* commands. Like the builder parameters, these parameters also allow variable substitution. When parameters are specified for a part, they are used in place of the parameters specified for the builder. In other words, part parameters take precedence over builder parameters.

In addition, whenever parameters are specified for any part that is an output of a build event, they apply to all of the output of that build event. For example, if a build event has two outputs, *mypgm.exe* and *mypgm.map*, changing the part parameters to */Debug* for either of the two parts has the same result. The next time the build event is processed, the */Debug* parameter is used when invoking the build script that produces both *mypgm.exe* and *mypgm.map*.

You can also substitute the builder parameters by using the *\$(BUILDERPARMS)* variable. For example, you might use the following command:

```
teamc part -build mypgm.c -parameters "/Ti+ $(BUILDERPARMS)" ...
```

At build time, the parameters specified in the builder for *mypgm.c* will be substituted by *\$(BUILDERPARMS)*.

As parameters of the part -build command

The *part -build* command parameters are used to set the values of environment variables that can be retrieved by the build script and for substitution of any variables set in the builder or part parameters.

For example, if you issue a *part -build* command for *mypgm.exe*, you can specify *-parameters DEBUG=YES*, which can be picked up by both the compile and link build scripts and used to set compiler or linker flags accordingly.

TeamConnection comes with the following sample compile and link build scripts for C/C++, COBOL, and PL/I:

fhbcob2.cmd

For compiling with the IBM COBOL VisualSet for OS/2 compiler

fhbcob2l.cmd

For linking or compiling and linking with the IBM COBOL VisualSet for OS/2 compiler and linker

fhbocomp.cmd

For compiling with the IBM C Set++ compiler

fhbolib.cmd

For the IBM IMPLIB command

fhbolin2.cmd

For linking with the VisualAge for C++ for OS/2 ICC command

fhbolink.cmd

For linking with the IBM ILINK command

fhborc.cmd

For the IBM Resource Compiler for OS/2

fhbplbld.cmd

For compiling with the IBM OS/2 PL/I compiler

fhbpllnk.cmd

For linking with the IBM PL/I LINK386 linker

You can use these sample build scripts as templates and tailor them to your own needs.

7.2.2 For MVS

On MVS, a build script can be either a JCL, REXX, or CLIST file. Figure 86 shows an example of a JCL for a C compile on MVS. For MVS (if using JCLs), a build script is a text file that contains JCL statements with additional TeamConnection syntax and substitutable variables. TeamConnection parses these statements and does the necessary allocations and program calls for a build.

Use an existing JCL fragment that is used for transforming input into output as a starting point for an MVS build script. For example, suppose you want to create a builder that compiles a C source file into an OBJECT file, using IBM's C/370 compiler. You probably already have JCL that can be submitted as a batch job that does this.

When you create a build script for the MVS environment, you specify JCL statements with additional TeamConnection syntax. This build script is parsed by the build processor. From the parsed results, TeamConnection allocates the specified ddnames and data sets; it then determines and executes the programs dynamically. The MVS build processor uses the specialized TeamConnection syntax in the JCL to determine where to store the parts involved in an MVS build.

```

+-----+
|
|  /*-----
|  /* PROGRAM:  FHBMCM
|  /* IBM C/370 compile for MVS Sample TeamConnection Build Script
|  /* Compile Only
|  /*-----
|  /*TEST1 EXEC EDCC,
|  //          CPARM='&TCPARM'
|  /*
|  //COMPILE.SYSIN    DD TCEXT=(C, CPP), DISP=(NEW, DELETE),
|  //    SPACE=(32000, (30, 10)), UNIT=VIO,
|  //    DCB=(RECFM=VB, LRECL=100, BLKSIZE=3200)
|  //COMPILE.USERLIB  DD TCEXT=(H, HPP), DISP=(NEW, DELETE),
|  //    SPACE=(32000, (30, 30, 30)), UNIT=VIO,
|  //    DCB=(RECFM=VB, LRECL=100, BLKSIZE=3200)
|  //COMPILE.SYSPUNCH DD TCEXT=OBJ, DISP=(NEW, DELETE),
|  //    UNIT=VIO, SPACE=(32000, (30, 10)),
|  //    DCB=(RECFM=FB, LRECL=80, BLKSIZE=3200)
|  //COMPILE.SYSLIN   DD SYSOUT=*
|  //
|
+-----+

```

Figure 86. MVS JCL for C Compile

Subtopics

7.2.2.1 Extended JCL Syntax

7.2.2.2 Passing Parameters

7.2.2.3 Sample Build Scripts

7.2.2.1 Extended JCL Syntax

In Figure 86 in topic 7.2.2 the experienced MVS user might recognize that TeamConnection has extended the existing JCL syntax. The extended syntax tells the TeamConnection build processor where to put the input, where to get the output, and where to get messages from the translators after an MVS build.

To direct input, output, and messages, add **TCEXT=xxx** to the data set attributes defined to a ddname, where xxx is one of the following:

- ☐ The base name extension from the TeamConnection part--for example, **TCEXT=H**, where *H* is the extension from, for example, *AFTMVS.H*
- ☐ One or more base name extensions from TeamConnection parts, surrounded by parentheses--for example, **TCEXT=(H,HPP)**, where *H* is an extension from *AFTMVS.H* or *HPP* is an extension from *SERATA.HPP*.
- ☐ The **TCOUT** string, which declares that the contents of the data set assigned to the ddname will be sent back to TeamConnection. You can view this information in one of the following ways:
 - On an OS/2 command line, by typing *teamc part name -viewmsg* and pressing Enter
 - By selecting **Part > View > View build message** from the **Actions** pull-down menu in the **TeamConnection - Tasks** window.

When you add the *TCEXT* attribute to a ddname specification, you must also specify the following attributes to allocate the data set through dynamic allocations:

- ☐ **SPACE**
- ☐ **UNIT**
- ☐ **DCB**, which includes the **LRECL**, **BLKSIZE**, and **RECFM** attributes

The **UNIT** attribute defaults to **VIO** unless the **-U** parameter is specified when the MVS build processor is started.

For translation messages, you can allocate a data set to the **TC\$LIST** ddname and specify the attributes yourself. Otherwise, the build processor allocates this data set with the following attributes by default:

```
//TC$LIST DD  DCB=(RECFM=VB,LRECL=255,BLKSIZE=32640),
//  SPACE=(CYL,(2,1)),DISP=(NEW,DELETE),UNIT=VIO
```

7.2.2.2 *Passing Parameters*

TeamConnection also allows you to pass parameters to an MVS build script. To pass parameters to your build script, specify them in the *Parameters* attribute of the builder. The parameters are passed to MVS through the combination of the **PARM** keyword parameter on an EXEC card and the **&TCPARM** variable.

Note: Do not use single or double quotes in the *Parameters* attribute of the MVS builder definition. This rule follows standard JCL syntax for parameter substitution in the *PARM* keyword parameter of an EXEC statement.

You can use the following TeamConnection variables in writing MVS build scripts:

&TCPARM

This variable is substituted with any parameters passed along by the build process.

&TCINPUT

This variable is used for in-stream data. For each build input, the line where &TCINPUT appears is duplicated, and the &TCINPUT variable is substituted with the input name.

&TCOUTPUT

This variable is used for in-stream data. For each build output, the line where &TCOUTPUT appears is duplicated, and the &TCOUTPUT variable is substituted with the output name.

&TCWORKAREA

The name of the work area in which the build is being performed

&TCRELEAS

The name of the release in which the build is being performed

Note: The *&TCINPUT* and *&TCOUTPUT* substitutable variables have limited scope in the MVS build scripts and should be used only within the in-stream data.

You can define other variables by specifying a value for *Parameters* when you start a build. These variables are set in the *parameters* string (see Figure 90 in topic 7.5) passed to the build script. They can also be used for variable substitution within MVS build scripts; this variable substitution works similarly to JCL variable substitution.

7.2.2.3 Sample Build Scripts

TeamConnection comes with sample compile and link build script JCLs for Assembler, C/370, COBOL, and PL/I. The following build script JCLs come with the TeamConnection product:

FHBCOBM.JCL

IBM COBOL compile for MVS sample build script JCL

FHBMC370.JCL

IBM C/370 compile for MVS sample build script JCL

FHBMASM.JCL

IBM Assembler for MVS sample build script JCL

FHBMC.JCL

IBM C/370 compile for MVS sample build script JCL

FHBMLINK.JCL

IBM Linkage Editor for MVS sample build script JCL

FHBMPLI.JCL

IBM PL/I compile for MVS sample build script JCL

FHBPLKED.JCL

IBM C/370 prelinker for MVS sample build script JCL

You can use these sample build script JCLs as templates and tailor them to your own needs.

7.2.3 For Parsers

Build scripts are also used for **parsers**. TeamConnection comes with sample build scripts for parsing C, C++, COBOL, and PL/I. The following parser build scripts come with the TeamConnection product:

fhbcbprs.cmd

Sample COBOL parser build script. Parses both COBOL for OS/2 and COBOL for MVS source.

fhbopars.cmd

Sample C parser build script

fhbplprs.cmd

Sample IBM PL/I parser build script

You can use these sample build scripts as templates and tailor them to your own needs.

7.3 Organizing Build Pools and Build Environments

The **build pool** defines the set of **build agents** to be used to process the build request. The build agent polls the build pool to which it is assigned to check whether there are any **build events** that have to be processed. You specify the Pool parameter at build time.

The **build environment** is the string that matches a build agent with a build event. You use the Environment parameter to specify the target environment for the object you are modifying. The value that you specify for the Environment parameter must exactly match the environment value specified in a corresponding build agent.

You can use the Environment parameter to organize your **build servers** not only by platform, but also by language. You can, for example, have one build server for OS/2 COBOL compiles and one for OS/2 C or C++ compiles. By organizing the build servers by language (using the Environment parameter), you can do parallel builds. We recommend that you follow a naming convention for the Environment parameter and use values such as *os2_c* and *mvs_cobol*.

7.4 Organizing the Build Agents and Build Processors

The **build agent** and the **build processor** work as a pair. The pair is called a **build server**. In the first release of TeamConnection, the build agent and build processor use TCP/IP to communicate with each other. The build agent is always installed on an OS/2 machine, regardless of where the build processor is located. For MVS builds, the build processor, which is installed on the MVS machine, is connected to a build agent on an OS/2 machine. Figure 87 depicts TeamConnection's build topology.



Figure 87. Build Topology

If you install a build agent on a machine that does not have a TeamConnection family server, ObjectStore will then be installed with the build agent, and the build agent will use ObjectStore to access the family. You use the TeamConnection install program (see "Installing the TeamConnection Components" in topic 4.2) to install a build agent or build processor on OS/2. To install a build processor for MVS, you must first install the code on OS/2 and then upload it to MVS.

In "Planning Your Build Environment" in topic 3.8, we mention that you can do TeamConnection builds for *heterogeneous applications* and *heterogeneous environments*. We also mention that you can have different topologies for the build servers for distributed builds (different platforms, for example, MVS and OS/2). Parallel builds are also possible. You can achieve parallel builds by organizing the build servers not only by platform but also by *language*. If there are lots of compiles to perform, this could then be a way of splitting the work that has to be performed between different machines.

Subtopics

7.4.1 Starting Your Build Server

7.4.1 Starting Your Build Server

You have to start a build server before you can do any TeamConnection builds. Starting a build server is a separate operation from starting the TeamConnection family servers. You do not have to start any additional daemons for the family when you start a build server. Starting the build server means that you have to start a build processor and build agent pair.

Subtopics

7.4.1.1 Starting a Build Processor on OS/2

7.4.1.2 Starting a Build Processor on MVS

7.4.1.3 Starting a Build Agent

7.4.1.1 Starting a Build Processor on OS/2

To start an OS/2 build processor, do one of the following:

- Type the following command and press Enter:

```
teamproc -s socket_port [-c cache_location -n] [-k local_codepage]
```

Parameter	Description
-s <i>socket_port</i>	<p>Is the TCP/IP socket port. This name must match the <i>socket_port</i> specified in the teamag (used in starting a build agent, see "Starting a Build Agent" in topic 7.4.1.3). The <i>socket_port</i> value can have one of two formats:</p> <ul style="list-style-type: none"> □ If you have added a port for the build socket in your <i>services</i> file, you can use that. For example, assume your <i>services</i> file has the following line: <pre>bld2008 7469/tcp # Build processor on aldan</pre> <p>You can specify the socket port like this:</p> <pre>teamproc -s bld2008</pre> □ If you have not specified an alias name in the <i>services</i> file, you can specify <i>@address</i>. <i>address</i> is the port address. For example, you can specify the socket port as: <pre>teamproc -s @7469</pre>
-c <i>cache_location</i>	Is the name of the fully qualified name of the directory in which caching will take place for builds
-n	Specifies that the contents of the cache directory will be erased before the build processor started
-k <i>local_codepage</i>	Is the code page to which text data is converted for the build

- Create a copy of the TeamConnection build processor in the **TeamConnection Group** folder and use the **Settings** notebook to set the *socket port* value in the *Parameters* field of the **Program** page. Then double-click on the icon to start the build processor.

7.4.1.2 Starting a Build Processor on MVS

TeamConnection provides two JCLs that you can use to start the MVS build processor:

RUNPGM

Starts the build processor as a batch job. This *job* accepts build scripts in JCL format.

RUNPGMT

Starts the build processor as a TSO task. This *task* accepts build scripts in REXX or CLIST format.

In Figure 88 and Figure 89 we show both JCLs. The JCLs have been modified to suit our MVS environment. We have added the following JOB card:

```
//TRULSSOP JOB (999,POK),'Leif Trulsson',CLASS=A,MSGCLASS=T,
//              NOTIFY=TRULSSO,MSGLEVEL=(1,1)
```

We also modified the following JCL cards:

EXEC

We added a *TIME=NOLIMIT*, and we changed the parameters for the **PARM** keyword, so that the EXEC card looks like this:

```
//RUNPGM      EXEC PGM=TEAMPROC,PARM='-S @7470 -U VIO -T ',
//              REGION=4M,TIME=NOLIMIT
```

TIME=NOLIMIT

Means that the job will run until we cancel it. It will not be terminated because of default system settings.

-S @7470

Is the *socket* our MVS build processor will listen to. The port number is preceded by the at sign (@).

-U VIO

Indicates the default unit type for dynamic data set allocations. VIO is the default if this parameter is not specified.

-T

Turns tracing on. Use this parameter if you have problems with your build processor.

STEPLIB

The *STEPLIB* DD card points to our MVS run-time environment (library). In Figure 88, we only have one data set allocated to this DD card:

```
//STEPLIB DD DSN=TRULSSO.TEAMC.LOADLIB,DISP=SHR
```

But in Figure 89, we have several concatenated data sets:

```
//STEPLIB DD DSN=DSNA.DSNLOAD,DISP=SHR
//          DD DSN=TRULSSO.TEAMC.LOADLIB,DISP=SHR
//          DD DSN=EDC.V2R1M0.SEDCLINK,DISP=SHR
//          DD DSN=PLI.V2R3M0.SIBMLINK,DISP=SHR
//          DD DSN=COBOL.V1R4M0.COB2COMP,DISP=SHR
```

The reason for the extra concatenation of data sets is that this job is run as a TSO subtask, and it is appropriate to allocate the "expected" run-time libraries here.

TEAMPROC

This DD card points to the library where our build script JCLs are stored:

```
//TEAMPROC DD DSN=TRULSSO.TEAMPROC.JCL,DISP=SHR
```

EDCENV

This DD card points to the data set containing the *TC_CACHESIZELIMIT* and *TC_CACHEPRUNEMETHOD* environment variables:

```
//EDCENV      DD DSN=TRULSSO.TEAMC.ENV,DISP=SHR
```

To specify the size of the cache directory, you can set two environment variables in the *environment* file:

```
TC_CACHESIZELIMIT=n
TC_CACHEPRUNEMETHOD=value
```

TC_CACHESIZELIMIT= n

n is the maximum number of bytes to allow for the build processor's cache directory.

TC_CACHEPRUNEMETHOD= value

value defines the order in which parts are pruned if the cache directory reaches the value of TC_CACHESIZELIMIT. Valid values are:

SIZE

Largest parts are pruned first.

DATE

Least recently used parts are pruned first.

If you specify TC_CACHESIZELIMIT but not TC_CACHEPRUNEMETHOD, the default pruning method is by size.

```
//TRULSSOP JOB (999,POK),'Leif Trulsson',CLASS=A,MSGCLASS=T,
//          NOTIFY=TRULSSO,MSGLEVEL=(1,1)
//*RUNPGM  JOB YOUR-JOB-CARD-HERE
//*****
//* PROGRAM: RUNPGM
//* JCL to start TeamConnection MVS Build Server
//*****
//*
//*          IBM TeamConnection for OS/2
//*          Version 1 Release 0
//*
//*          5622-717
//* (C) Copyright, IBM Corp., 1995. All Rights Reserved.
//*          Licensed Materials - Property of IBM
//*
//*          US Government Users Restricted Rights
//* - Use, duplication or disclosure restricted by
//*          GSA ADP Schedule Contract with IBM Corp.
//*
//*          IBM is a registered trademark of
//*          International Business Machines Corporation
//*
//* DISCLAIMER OF WARRANTIES:
//* The following enclosed code is sample code created
//* by IBM Corporation. This sample code is not part of
//* any standard product and is provided to you solely
//* for the purpose of assisting you in the development
//* of your applications. The code is provided "AS IS",
//* without warranty of any kind. IBM shall not be
//* liable for any damages arising out of your use
//* of the sample code
//*
//*****
//* Some dataset names may need to be modified
//* according to your system's customization
//*****
//RUNPGM EXEC PGM=TEAMPROC,PARM='-S @7470 -U VIO -T ',
//          REGION=4M,TIME=NOLIMIT
//STEPLIB DD DSN=TRULSSO.TEAMC.LOADLIB,DISP=SHR
//TEAMPROC DD DSN=TRULSSO.TEAMPROC.JCL,DISP=SHR
//EDCENV DD DSN=TRULSSO.TEAMC.ENV,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//
```

Figure 88. The RUNPGM JCL

In Figure 89 we actually start the build processor by a CALL statement in the SYSTSIN DD card:

```
CALL 'TRULSSO.TEAMC.LOADLIB(TEAMPROC)' '-S @7040 -U SYSDA '
```



```
//TRULSSOP JOB (999,POK),'Leif Trulsson',CLASS=A,MSGCLASS=T,
//      NOTIFY=TRULSSO,MSGLEVEL=(1,1)
//*****
//* PROGRAM:  RUNPGMT                                     */
//* JCL to start TeamConnection MVS Build Server under    */
//* a TSO environment.  Primary use is intended for       */
//* VisualGen support.                                   */
//*****
//*                                                     */
//*          IBM TeamConnection for OS/2                  */
//*          Version 1 Release 0                          */
//*                                                     */
//*          5622-717                                     */
//* (C) Copyright, IBM Corp., 1995. All Rights Reserved. */
//*          Licensed Materials - Property of IBM         */
//*                                                     */
//*          US Government Users Restricted Rights        */
//* - Use, duplication or disclosure restricted by        */
//*          GSA ADP Schedule Contract with IBM Corp.     */
//*                                                     */
//*          IBM is a registered trademark of             */
//*          International Business Machines Corporation   */
//*                                                     */
//* DISCLAIMER OF WARRANTIES:                            */
//* The following enclosed code is sample code created   */
//* by IBM Corporation.  This sample code is not part of */
//* any standard product and is provided to you solely   */
//* for the purpose of assisting you in the development  */
//* of your applications.  The code is provided "AS IS",  */
//* without warranty of any kind. IBM shall not be       */
//* liable for any damages arising out of your use       */
//* of the sample code                                   */
//*                                                     */
//*****
//* Some dataset names may need to be modified          */
//* according to your system's customization            */
//*****
//RUNPGMT EXEC PGM=IKJEFT01,REGION=6M,DYNAMNBR=30,TIME=NOLIMIT
//STEPLIB DD DSN=DSNA.DSNLOAD,DISP=SHR
//      DD DSN=TRULSSO.TEAMC.LOADLIB,DISP=SHR
//      DD DSN=EDC.V2R1M0.SEDCLINK,DISP=SHR
//      DD DSN=PLI.V2R3M0.SIBMLINK,DISP=SHR
//      DD DSN=COBOL.V1R4M0.COB2COMP,DISP=SHR
//TEAMPROC DD DSN=TRULSSO.TEAMPROC.JCL,DISP=SHR
//EDCENV   DD DSN=TRULSSO.TEAMC.ENV,DISP=SHR
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
//      CALL 'TRULSSO.TEAMC.LOADLIB(TEAMPROC)' '-S @7040 -U SYSDA '
//
```

Figure 89. The RUNPGMT JCL

You start the MVS build processor by submitting either of the two JCLs (see Figure 88 and Figure 89). You can of course have both of the build processors running, but then you have to have a unique port number for each of them.

When an MVS build processor runs a build script, it creates a cache data set for each unique file extension type associated with a **TCEXT** tag that it encounters. When the build completes successfully, the data sets created for the build are deleted, and their contents are moved to the cache data sets.

Each cache data set is created by using the attributes specified in the DD statement associated with the **TCEXT** attribute. If attributes are not specified, these are the defaults used for creating a cache data set:

```
DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
UNIT=VIO
SPACE=(CYL,(30,30,1000))
```

The cache data sets are associated with the **TEAMPROC** address space, and they are deleted when the build processor is stopped.

7.4.1.3 Starting a Build Agent

A build agent handles access to the family data on behalf of the build processor. Each build agent is connected through a TCP/IP connection to one and only one build processor.

There can be any number of build agents. When you start the build agents, you assign them to build pools (see "Organizing Build Pools and Build Environments" in topic 7.3). In a single pool, some agents might be connected to OS/2 build processors and others to MVS build processors. The *environment* specified in the builder for a particular build event determines which agents in the pool are available to it.

A pool is formed when you start the first build agent that specifies its name. There is no limit to the number of agents that you assign to a build pool.

To start a build agent, do one of the following:

- From the command line, type the following and press Enter:

```
teamagt -s socket_port -f familyname -p poolname -e environment
        [-k local_codepage]
```

Parameter	Description
-s <i>socket_port</i>	Is the TCP/IP socket port. This name must match the <i>socket_port</i> specified in the teampr used to start the corresponding build processor (see "Starting a Build Processor on OS/2" in topic 7.4.1.1 and "Starting a Build Processor on MVS" in topic 7.4.1.2). The <i>socket_port</i> have one of two formats: <ul style="list-style-type: none"> □ If you have added a port for the build socket in your <i>services</i> file, you can use that. For example, assume your <i>services</i> file has the following line: <pre>bld2008 7469/tcp # Build processor on aldan</pre> You can then specify the socket port like this: <pre>teamagt -s bld2008 -f fam2008 -p pool1 -e os2_c</pre> □ If you have not specified an alias name in the <i>services</i> file, you can specify <i>@hostname:port</i> where <i>hostname</i> is the name of the host where the build processor is installed, and <i>port</i> is the port address. For example, you can specify the socket port as follows: <pre>teamagt -s @bld2008@7469 -f fam2008 -p pool1 -e os2_c</pre>
-f <i>familyname</i>	Is the name of the TeamConnection family. If the build agent is installed on a different machine from the family, the fully qualified name is <i>hostname:dbpath\family_name.tcd</i> , where: <ul style="list-style-type: none"> □ <i>hostname</i> is the name of the machine where the family resides. □ <i>dbpath</i> is the drive and path for the family. □ <i>family_name</i> is the name of the TeamConnection family.
-p <i>poolname</i>	Is the name of the build agent pool
-e <i>environmentname</i>	Specifies the <i>environment</i> for which you are building (see "Organizing Build Pools and Build Environments" in topic 7.3). The value you specify here must exactly match the <i>environment</i> for a builder in order for the builder to use this build agent. This value is case sensitive.
-k <i>local_codepage</i>	Is the code page for data stored in TeamConnection

- Create a copy of the TeamConnection build agent in the **TeamConnection Group** folder and use the **Settings** notebook to set the **-a** *parameter_file* value in the *Parameters* field of the **Program** page. Notice that the TeamConnection build agent in the **TeamConnection Group** folder uses the *teamcd.exe* to start the build agents.

Usually the **teamcd** command is used to start the family server. The command syntax for *teamcd* is as follows:

```
teamcd -m -d -a <agents> -p <procs> -n <mailexit> family clones
```

Parameter	Description
-m	Maintenance mode (for example, no database updates allowed)
-d	Debug mode. No subprocesses are started, and instead the clients are served from the main process. This allows you to use IBM presentation manager debugger (IPMD) on the main process server.

TeamConnection at Large Starting a Build Agent

	If you specify debug mode, the <i>agents</i> , <i>mailexit</i> , <i>procs</i> , and <i>clones</i> parameters are ignored.	
-a agents	The name of a file containing a description of the build agents you want to start (see below). In lieu of this parameter, you can set the value using the <code>TC_BUILD_AGENTS_FILE</code> environment variable.	
-p procs	The name of a file containing a description of the build processors you want to start (see below). In lieu of this parameter, you can set the value using the <code>TC_BUILD_PROCESSORS_FILE</code> environment variable.	
-n	mailexit	Is the name of an executable that the notify daemon should use to process mail notifications. In lieu of this parameter, you can set this value using the <code>TC_NOTIFY_DAEMON</code> environment variable.
<mailexit>		
family	family	The name of the family
clones	clones	The number of family server subprocesses to process client requests. Default value is 1, which means that no client subprocesses are started.

If you use the **teamcd** command to start your build agents, you have to create a parameter file containing the necessary information to start the build agents. The parameter file contains one line per build agent. The syntax is as follows (lines beginning with **#** are treated as comments):

```
-s <socket_port> -e <environment> -k <local_codepage> -p <pool>
```

where the parameters are the same as for the **teamagnt** command.

If you prefer to use the **teamagnt** command instead of the **teamcd** command to start the build agent, use the **Settings** notebook to change the name of the program in the *Path and file name* field from *teamcd.exe* to *teamagnt.exe*. Then set the **-s socket_port -f familyname -p poolname -e environment [-k local_codepage]** value in the **Parameters** field of the **Program** page.

Double-click on the icon to start the build agent.

7.5 Creating a Builder

A **builder** is a TeamConnection object that describes how to perform a build event, that is, how to translate input managed objects into output managed objects. For example, one builder might know how to transform C++ source code into object code. Another builder might know how to transform object code into an executable. The builder identifies the environment where the transform will execute, passes parameters to and from its build script, defines the basic rules of successful completion, and invokes a build script.

Use the **Create Builders** window (see Figure 90) to define a builder with a build object. You must specify a number of attributes when you create a builder. Together with the contents of the build script and the tools you use (the compilers, linkers, and other tools), the following attributes define how a transformation takes place:

Name

The name of the builder must be unique within a release. It can be anything you want; we recommend that you establish and follow a meaningful naming convention. An example of a builder name is *C-compile*.

Release

This is the name of the release that contains the builder. Builders are release-specific objects. They are not versions within a release. Therefore you can have only one version of a builder at any time in a release.

To use the builder from a previous release, you can link to a part that uses it in the previous release. This action copies the builder to the new release. Otherwise, you must create it again in the new release.

Script, File type, and Source file

These fields work together to define the build script that the builder invokes to accomplish the transformation. (The **File type** field on the GUI corresponds to **-text**, **-binary**, and **-none** in the command. The **Source file** field on the GUI corresponds to the **-from** attribute in the command.)

If this is an MVS build script, you must first create a separate OS/2 file containing the build script. All MVS build scripts must be written with the JCL statements and the TeamConnection syntax described in "For MVS" in topic 7.2.2. You can store the build script one of two ways:

- ☐ As part of the builder

Specify the fully qualified path name of your build script file as the source file. Specify the file type as text. When the builder is created, this source file is stored as part of it in the TeamConnection database.

During a build, the build processor creates and runs a local version of this file. Specify the name you want for this local file in the Script field. For example, you might specify these values:

File type

text

Script

fhbc

Source file

D:\build\script\fhbc.jcl

When this builder is created, the contents of *D:\build\script\fhbc.jcl* are stored in the builder. When this builder is invoked, TeamConnection creates a file named *FHBC* in the data set referenced by the *TEAMPROC* ddname, writes the build script into this file, and then runs it.

- ☐ On MVS

Create the build script file and place it in the data set allocated to the *TEAMPROC* ddname. When you do this, specify the following attributes:

File type

none

Script

link

Do not specify a source file.

Environment

This is the name of the environment supported by the builder, such as OS/2 or MVS. The value that you specify here must exactly match the environment value specified in a corresponding build agent. Again, we recommend that you follow a naming convention for this attribute and use values such as *os2_c* and *mvs_cobol*.

Condition and Value

Together, these two attributes make up a Boolean expression that defines the criteria used to decide whether a specific build event was successfully accomplished when evaluated against the value returned by the build script. Valid comparison operators are as follows:

== or EQ	Equals
< or LT	Less than
<= or LE	Less than or equals
> or GT	Greater than
>= or GE	Greater than or equals
<> or NE	Not equal to

Value can be any positive integer. An example of a Boolean expression formed from these two attributes is **return_value LE 4**; that is, the build event is considered a success if the build script returns a value less than or equal to 4.

Parameters

This is a string passed to the build script as its argument. For example, for a builder used for CSet/2 compiling, you might specify a parameter string of */Ss /Ge-*, and for an MVS builder used for linking load modules, you might specify a parameter string of *list,test*.

Timeout

This attribute specifies the number of minutes that a build server will wait for an invoked build script to return before concluding that an error has occurred and stopping the build event. If a build event is stopped, it is queued again to be processed by the next available build agent.

The following fields and push buttons are available in the **Create Builder** window (see Figure 90):

Builder

Type the name of the builder you want to create.

Release

Type the name of the release under which the parts you are building are stored in TeamConnection.

Script

Type the name of the build script part. For example, in OS/2, this part is an executable or command file that specifies the steps a build operation should take. It could be a compiler or linker or the name of a CMD file you have written.

Environment

Type the target environment for the object you are building, such as OS/2, MVS, or AIX.

Comparison operator

Type the comparison operator that determines whether a build is successful. This field is used in conjunction with the RC value field. Valid comparison operators are:

== or EQ	Equals
< or LT	Less than
<= or LE	Less than or equals
> or GT	Greater than
>= or GE	Greater than or equals
<> or NE	Not equal to

RC value

Type the numeric value that TeamConnection uses along with the comparison operator to determine whether a build is successful.

For example, suppose you typed 0 in this field and EQ in the Comparison operator field. When the build is started, if the value returned by the script file is equal to 0, TeamConnection indicates that the build is successful.

File type

Select the designation that indicates the type of data a file contains. Select the file type of the files from one of the following:

Text

Specifies a part with ASCII- or EBCDIC-encoded data

Binary

Specifies a part other than ASCII- or EBCDIC-encoded data. Examples of such parts are graphical, object code, or executable parts.

None

Specifies a placeholder object. There is no part associated with this object.

Source file

Type the current location of the script file. For example, if the script file is called *compiler.exe* and is located on your C drive in a directory called *tools*, you enter *c:\tools\compiler.exe* in this field.

Select

Displays the **Select Source File** window. From this window, you can select a part name from your workstation. The part you select appears in the Source file field.

Parameters

Type the parameters that are required by the executable file.

Timeout

Type the number of minutes before TeamConnection determines that a build has failed.

OK

Processes the information that you typed and closes the window

Apply

Processes the information that you typed and leaves the window open

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

Create Builder

Builder: c370

Release: AFT-MVS

Script: fhbmc

Environment: mvs

Comparison operator: <=

RC value: 4

File type: ☒ Text ☐ Binary ☐ None

Source file: D:\TEAMC\BIN\fhbmc.jcl

Parameters:

Timeout: 1440

Figure 90. Create Builder Window

Command Line Interface Syntax

The following is the command line interface syntax for creating a builder:

```
teamc builder -create Name -condition RCExpression
             -environment Name
             -from ScriptFilespec**
             -script Name
             -value RCValue -release Name
             -family Name
             [-text* | -binary | -none]
             [-parameters Parameters]
             [-timeout Number] [-become UserName]
             [-verbose]
```

Attribute Flag and Argument	Description
-create Name	Creates a new builder
-condition RCExpression	Together with the -value flag, makes up a Boolean expression that defines the criteria used to decide whether a specific build event was successfully accomplished when evaluated against the value returned

by the build script.
The values allowed
for this flag are:

EQ or ==	Equals
LT or <	Less than
LE or <=	Less than or equals
GT or >	Greater than
GE or >=	Greater than or equals
NE or !=	Not equal to

-environment
Name Specifies the target environment for which this builder builds. The environment string should match the string specified on the build agent that you want to handle the build events performed by this builder.

-from
ScriptFilespec Specifies the location of the script file whose contents are to be stored in the database

-script *Name* Specifies the name of the build script. It can also identify the translator when the **-none** flag is specified.

-value
RCValue Together with the **-condition** flag, makes up a Boolean expression that defines the criteria used to decide whether a specific build event was successfully accomplished when evaluated against the value returned by the build script. The **-value** can be any positive integer. An example of a Boolean expression formed from these two attributes is *return_value LE 4*, indicating that the build event is considered a success if the build script returns a value less than or equal to 4.

Attribute Flag and Argument	Description
-----------------------------------	-------------

-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
--------------------------------	---

-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-text	Specifies that the build script is a text file
-binary	Specifies that the build script is a binary file
-none	Specifies that the build script is not stored in TeamConnection
-parameters <i>Parameters</i>	Specifies the parameters passed to the build script
-timeout <i>Number</i>	Specifies the amount of time that the build processor waits for a build script to complete before assuming that a failure has occurred. The default is 1440 minutes.
-become <i>Name</i>	Identifies the use ID from which you want to issue TeamConnection commands if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME)
-verbose	Specifies that you want to receive a confirmation message after you issue this command

Using the above syntax, our builder create would then look like this::

```
teamc Builder -create c370 -release AFT-MVS -script fhbmc
              -environment mvs -condition "<=" -value 4
              -from D:\TEAMC\BIN\fhbmc.jcl -text
```

7.6 Creating a Parser

A **parser** is a tool that knows how to read a source file and report back a list of dependencies of that source file. A parser simplifies the job of defining a build tree. It frees you from having to know which dependencies a managed object has on other managed objects. For example, a C parser knows how to read a C source code file and report back a list of files included by the source file. If dependencies are found during the parse that are not reflected in the build tree, the build tree is automatically updated. A set of sample language parsers and build scripts is provided with TeamConnection.

Use the **Create Parser** window (see Figure 91) to associate a parser with a build object.

The **Create Parser** window contains the following fields and push buttons:

Parser	Type the name of the parser
Release	Type the name of the release
Command	Type the path and part name of the parser command
Include	Type the series of paths, separated by colons and semicolons, used to search for dependencies in the database. For example: source\include;.;
OK	Processes the information that you typed and closes the window
Apply	Processes the information that you typed and leaves the window open
Cancel	Does not process the information that you typed and closes the window
Import	Imports text into the field that has cursor focus
Help	Displays help information about the window

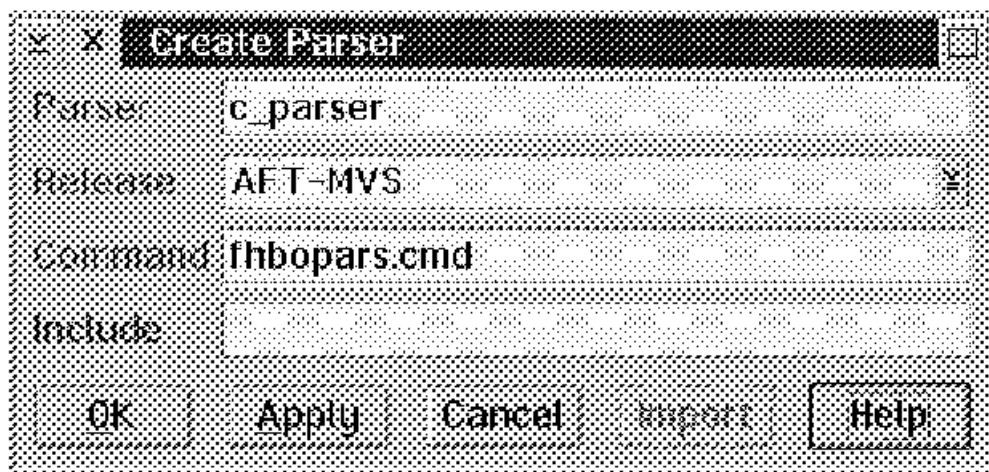


Figure 91. Create Parser Window

```

-----
|
| Command Line Interface Syntax
|
| The following is the command line interface syntax for creating a
| parser:
|
|
| teamc parser -create Name -command Name -release Name -family Nam
|
|

```

```
[-include Paths]  
[-become UserName] [-verbose]
```

Attribute Flag and Argument	Description
-create <i>Name</i>	Creates a new parser
-command <i>Name</i>	Specifies the command file you want to associate with the parser. This can be an .exe , .com , .cmd , or .bat file. The executable file must be in the execution path of the TeamConnection family server.
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-include <i>Paths</i>	<p>Specifies a concatenated set of paths that define where the parser looks for parts when processing the set of dependencies returned from the command file. There are two types of dependencies:</p> <ul style="list-style-type: none">□ A dependency in which the file is stored in the TeamConnection database. For example, <i>hello.c</i> includes <i>hello.h</i>, and both files are stored in the TeamConnection database. During a build, these dependencies must be extracted to a path accessible by the build processor.□ A dependency on a file that is not stored in the TeamConnection database. An example of such a dependency is <i>stdio.h</i>, which is typically stored in a compiler's

TeamConnection at Large

Creating a Parser

include path and
not in the
TeamConnection
database.

Attribute Flag and Argument	Description
-----------------------------------	-------------

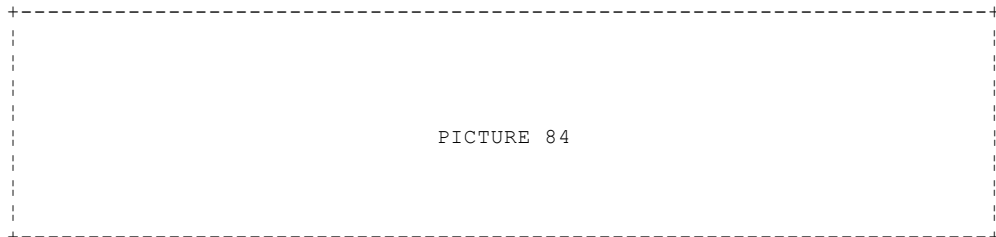
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME)
-verbose	Specifies that you want to receive a confirmation message after you issue this command

Using the above syntax, our parser create would then look like this:

```
teamc Parser -create c_parser -release AFT-MVS -command fhbopars.cmd
```

7.7 Creating a Build Tree

A **build tree** (see Figure 92) is a structure that graphically defines how the managed objects are built together. The build tree is a complete description of the dependencies that managed objects have on one another and the steps required to build the release. Each executable step in the build tree is a **build event**. For example, the compilation of C source into object code is a build event.



PICTURE 84

Figure 92. Build Tree

Use the **TeamConnection - BuildView** window (see Figure 93) to see the relationship between parts in a build tree structure. For example, you can see whether a part is a dependent of another part during a build operation, or you can see a list of all of the parts that are associated with a particular builder or parser.

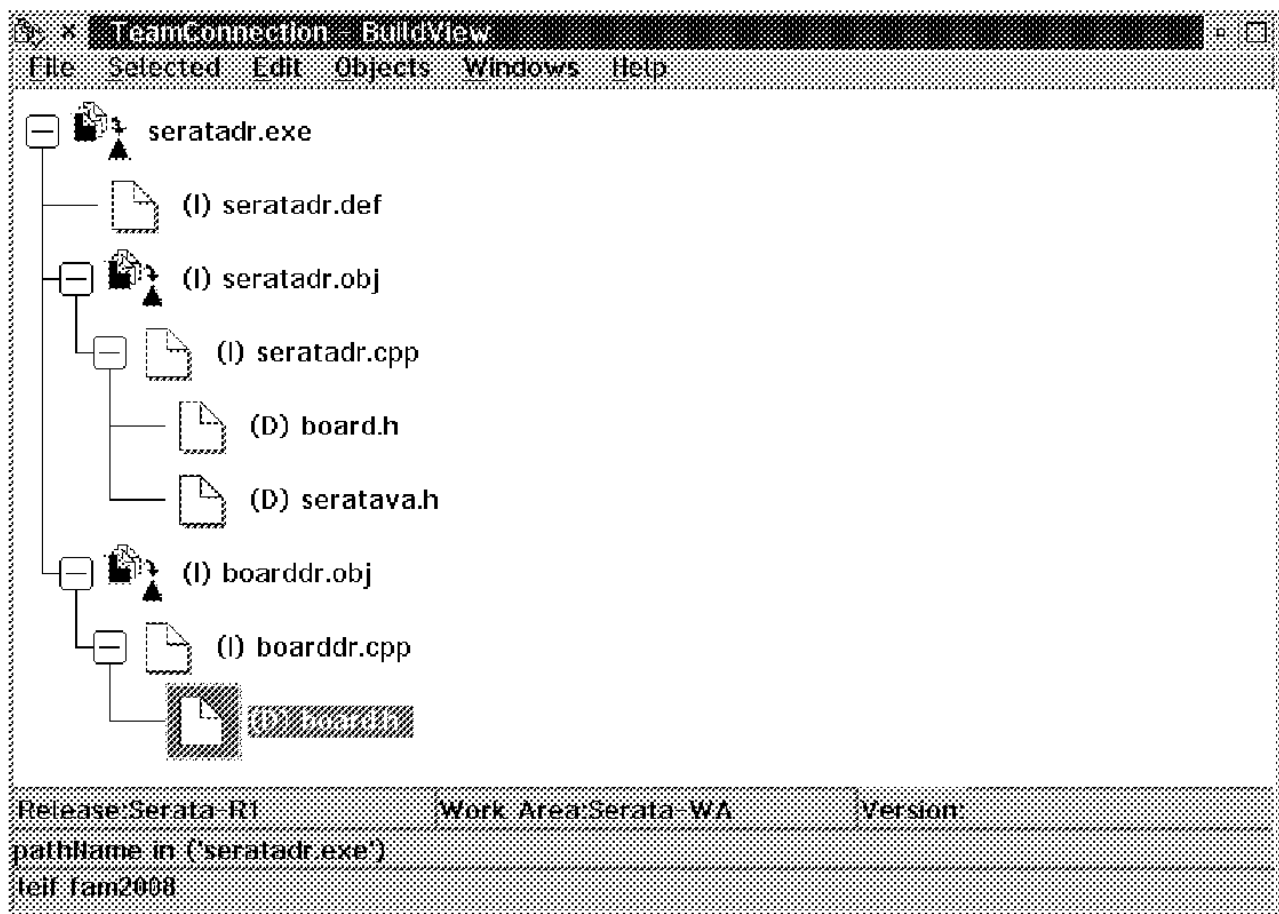


Figure 93. TeamConnection - BuildView Window

The following is an example of a build tree:

```
hello.exe
|
|_ hello.map (output relationship)
|
|_ hello.obj (input relationship)
|
|_ hello.c (input relationship)
|
|_ hello.h (dependent relationship)
```

This example illustrates the following:

- The *hello.map* part has an output relationship to *hello.exe*. Thus a

build of *hello.exe* will also generate and store *hello.map* in the library.

- The *hello.obj* part has an input relationship to *hello.exe*. Thus *hello.obj* is input into the build event that builds *hello.exe* and *hello.map*.
- The *hello.c* part has an input relationship to *hello.obj*. Thus *hello.c* is input into the build event that builds *hello.obj*.
- The *hello.h* part has a dependent relationship on *hello.obj*. Thus *hello.c* has a dependency on *hello.h*. If *hello.h* is updated, *hello.obj*, and *hello.exe* are out-of-date. Also, during the build of *hello.obj*, *hello.h* is extracted from the library and used by the build server. Without this relationship, the compile will fail because it will not find *hello.h*.

Creating the Build Tree

Creating and maintaining the build tree is part of the **part** command. In Appendix B, "Build Tree Aid" in topic B.0, we show you the command line syntax to use to create and maintain your build trees. We also provide a table that you can use as an aid in your build tree creation.

You can use both the GUI and the command line syntax to create your build trees. If you decide to use the GUI, you create and maintain your build trees from either the **TeamConnection - Parts** window or the **TeamConnection - BuildView** window and by using the **Connect... (=)** and **Disconnect... (=)** actions from either the **Selected** menu item or the pop-up menu.

7.8 Using a NULL Builder

A **NULL builder** is commonly used as a collector object. It is what you would call a *pseudotarget* in *MAKE* terminology. You can use a NULL builder to build complex applications that consist of many different build targets that must be built in one single job (see Figure 94).

The input to a NULL builder is not transferred to the build processor. The NULL builder build event is simply marked as successful when it is encountered by the build agent.

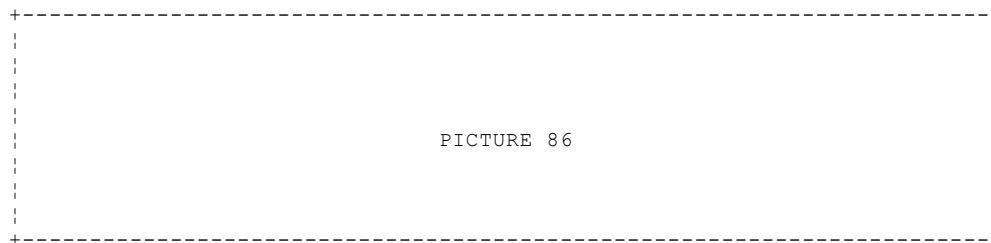


Figure 94. Application Build Using a NULL Builder

A NULL builder is created by using the **builder -create** command (see "Creating a Builder" in topic 7.5) and the **-script** and **-none** attributes. Make sure you specify a type of **-none** and either **"** or **"NULL"** for the *Name* argument in the **-script** attribute.

Here is an example:

```
teamc builder -create MyNullBuilder -release Serata-R1 -script NULL
            -environment os2 -condition "==" -value 0 -none
```

The **-value** and **-condition** parameters do not have any effect for NULL builders, but TeamConnection requires that you specify them in the **builder -create** command.

Now that you have a NULL builder created, you can use it as a *collector object* for an application. You create a *collector object* by using the **part -create** command:

```
teamc part -create SerataApplication -release Serata-R1 -builder MyNullBuilder
            -none -component Serata-C++ -workarea Serata-WA
```

Note: You can of course also use the GUI (see Figure 90 in topic 7.5) to create a NULL builder just like you create any other builder.

7.9 Building an Application

Once the build tree, builders, build scripts, and parsers are set up, TeamConnection has a reliable, repeatable build process that minimizes the time and resource required to build an application. When a build is requested, the build process determines out-of-date managed objects (objects that have been changed or whose inputs have been changed) within the build tree, parses any out-of-date objects to find any new dependencies, and uses the builders and build scripts to rebuild the out-of-date objects in the requested environment.

A build can be started for any output file. The appropriate build events for all necessary compiles and links are generated, and a build agent passes the information to an appropriate build processor for the target environment (see Figure 95).

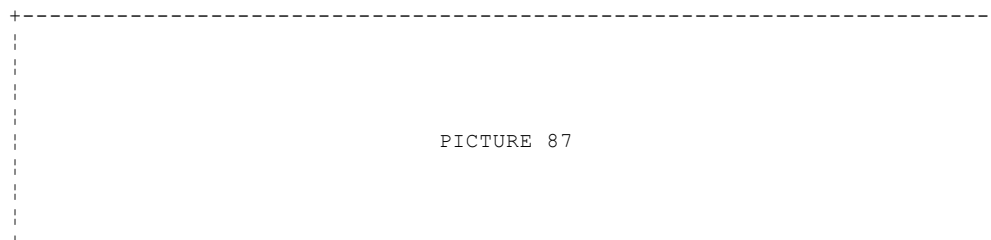


Figure 95. Application Build

You can start a build in many ways:

- ☐ Through **Actions > Parts > Build...** in the **TeamConnection - Tasks** window
- ☐ Through **Selected > Build...** in the **TeamConnection - Parts** window
- ☐ Through the **Build...** pop-up choice in the **TeamConnection - Parts** window
- ☐ Through **Selected > Build...** in the **TeamConnection - BuildView** window
- ☐ Through the **Build...** pop-up choice in the **TeamConnection - BuildView** window
- ☐ By using the **part -build** command from the command line or command file

You use the GUI when you want to start the build by hand. If you want to have a more automated build, for example, a build that starts at the same time every day, you would start the build from a REXX command file, for example.

Use the **Build Parts** window (see Figure 96) to build selected parts. You start a build against an output part with which you have associated a builder. A builder is a command part that you create that describes how to translate input parts to get the desired output. For example, if you wanted to use a particular compiler for a build operation, you would specify the name of the compiler when you created the builder. You can also associate a parser with a part so that include dependencies can be determined during the course of the build. The build function enables you to:

- ☐ Track time stamps of input and output so that it builds only those parts that are out-of-date
- ☐ Force a build regardless of the time stamps
- ☐ Spread the build over multiple machines running at the same time
- ☐ Group build servers into classes or pools so that you can route work to a specific set of servers

The following fields and push buttons are available in the **Build Parts** window:

Path name

Type the path name of the part you want to build. The path name can include both the directory name and base name.

Type

Type the part type. For example, *File* is a part type. Parts with a part type of *File* can exist on the workstation. If you do not specify a part type, this field defaults to *File*. Depending on your development environment, you may have parts that have a part type other than *File*.

Release

Type the name of the release with which the parts are associated.

Work area

Type the name of the work area with which the parts are associated.

Pool

Type the name of the build server or group of build servers that will process the build. You can get this name from your family administrator.

Build mode

Select one of the following radio buttons to specify the type of build:

Normal

Builds only the parts that are out-of-date.
Processing stops after the first error is returned.

Forced

Builds all parts, even if they are out-of-date.
Processing stops after the first error is returned.

Unconditional

Builds only parts that are out-of-date but continues processing even if errors are returned. Note that output is not rebuilt for input that has failed.

Report

Gives a preview of what would be built if you invoked a build. The report identifies the steps that would occur without any translations taking place.

Parameters

Type the parameters associated with the build.

Detail file

Type the name of the output file that TeamConnection uses to create a detail file of all build data. An output file can be an output only for a single *buildEvent*. Each *buildEvent* requires a separate and unique output file.

OK

Processes the information that you typed and closes the window

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

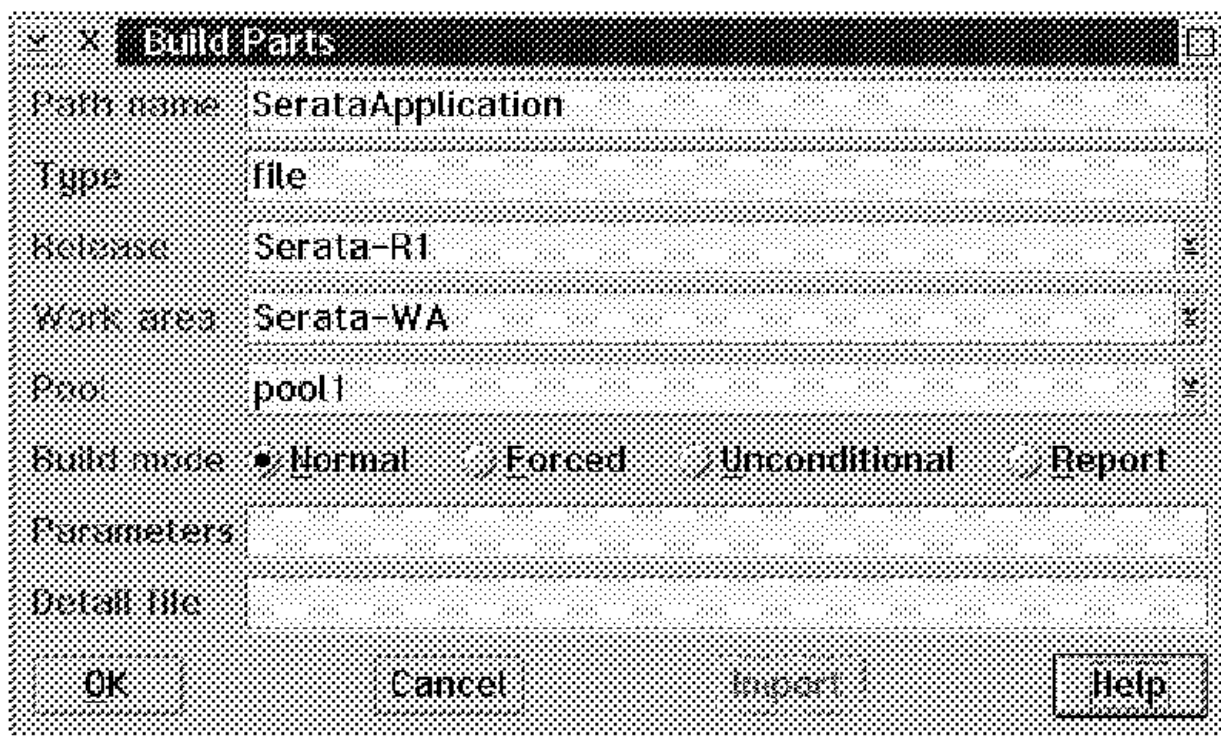


Figure 96. Build Parts Window

Use the **Build Progress** window (see Figure 97) to monitor the progress of a build you submit. When the family server starts returning information about the build, the **Build Progress** window appears. The *Path name*, *Type*, *Release*, and *Work areas* fields display the information you specified in the **Build Parts** window. The lower portion of the window is the progress log and displays the messages sent by the family server. By monitoring these messages, you can keep track of the progress of your build.

While the build is in progress, you can either stop or cancel the build. Select the **Stop** push button to disconnect TeamConnection from the family server, close the **Build Progress** window, and return control of the TeamConnection session to you. Although you can no longer access that build, it continues on the server until the build is either complete or fails. Select the **Cancel** push button to display the **Cancel Part Build** window. After you cancel the build, select the **Stop** push button to close the **Build Progress** window and disconnect from the family server.

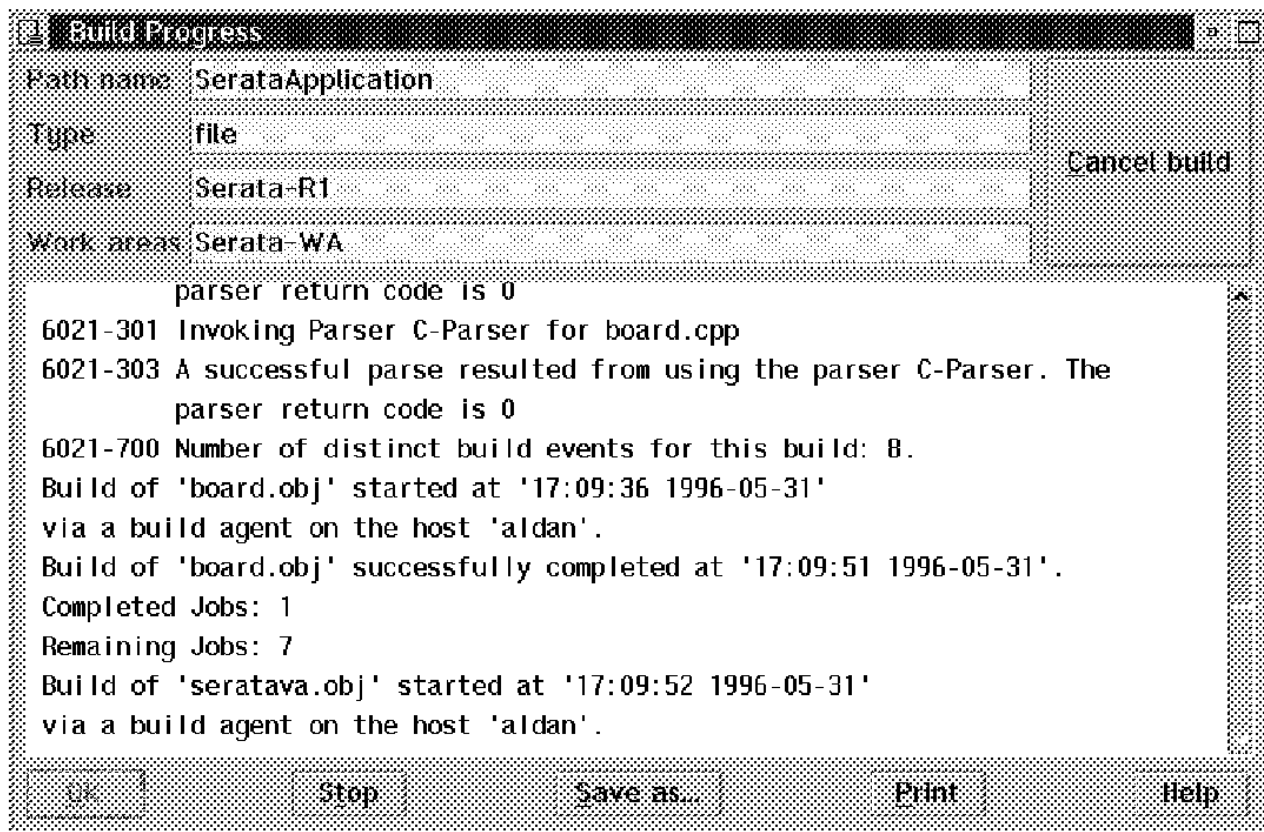


Figure 97. Build Progress Window

In Figure 97, the following lines indicate that the *C-Parser* has successfully parsed the *seratadr.cpp* source file:

```

6021-301 Invoking Parser C-Parser for seratadr.cpp
6021-303 A successful parse resulted from using the parser C-Parser. The
        parser return code is 0
  
```

This line tells you that the build function has located eight distinct build events. They could be compile, link, or any other build event, but there are eight of them.

```

6021-700 Number of distinct build events for this build: 8.
  
```

The next lines you see are:

```

Build of 'board.obj' started at '17:09:36 1996-05-31'
via a build agent on the host 'aldan'.
Build of 'board.obj' successfully completed at '17:09:51 1996-05-31'.
Completed Jobs: 1
Remaining Jobs: 7
  
```

Here the build function indicates that the build of the *board.obj* part started at 17:09:36 1996-05-31 and the build event was *picked up* by a build agent on the *aldan* machine. The build event was successful and completed at 17:09:51 1996-05-31. Furthermore, one job has completed, and

there are seven more to go.

Build completion is indicated in the **Build Progress** window (see Figure 98).

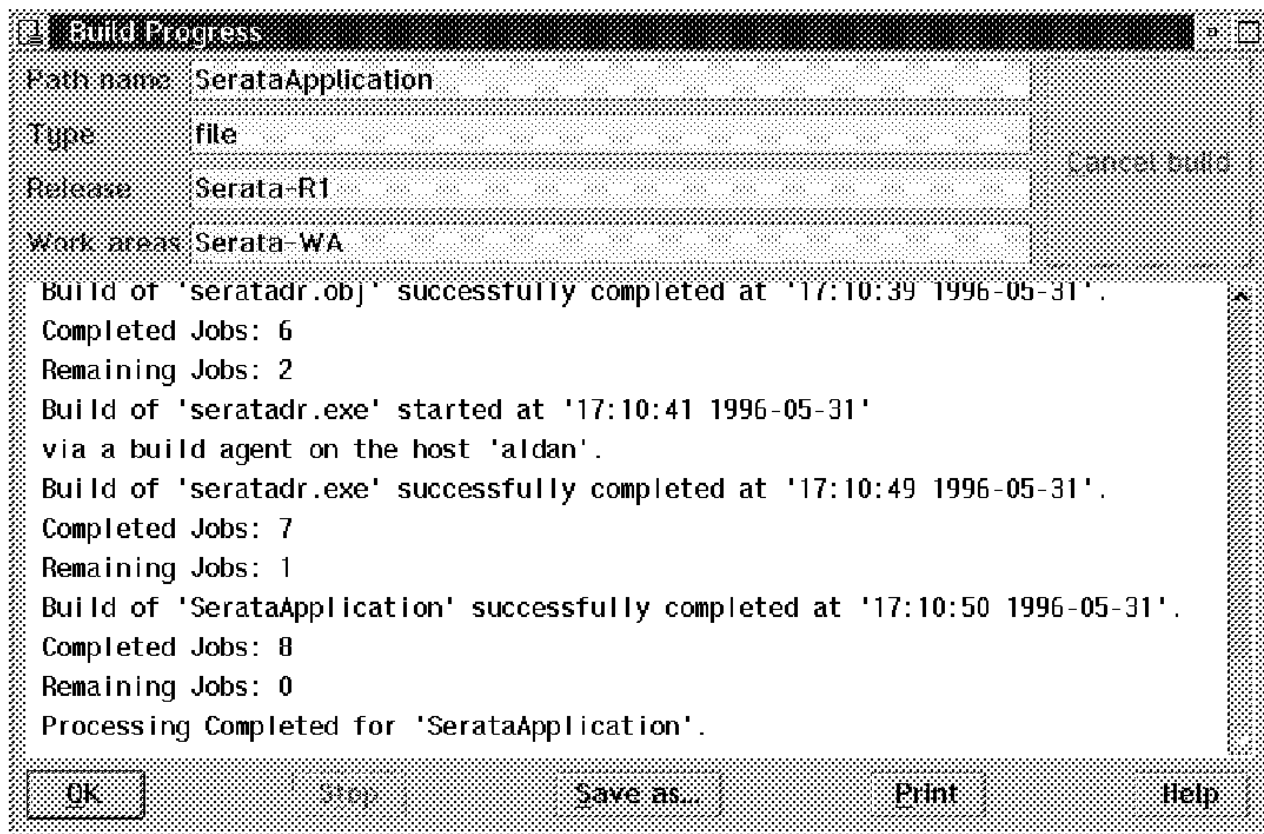


Figure 98. Build Progress Window after Completed Build

The last four lines in Figure 98 indicate that the build of the *SerataApplication* completed successfully at 17:10:51 1996-05-31, eight jobs completed, zero remain, and processing is complete.

Figure 99 shows the **TeamConnection - BuildView** window after the completed build. We only show a partially expanded build tree, but you can see that the icons for the build targets have changed from PICTURE 91 to PICTURE 92 (refer to Figure 93 in topic 7.7), indicating that the build has been successful.

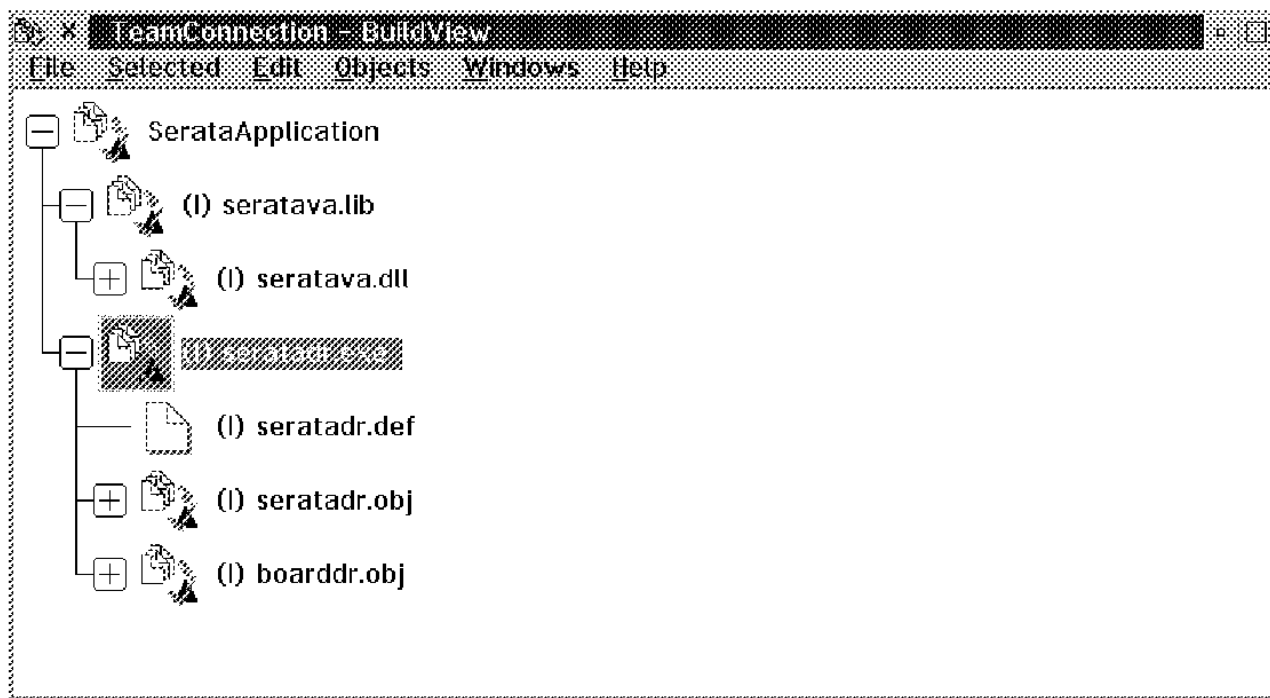


Figure 99. TeamConnection - BuildView Window after Completed Build

Use the **View > View build message...** from either the **Selected** menu item or the pop-up menu in either the **TeamConnection - Parts** window or the **TeamConnection - BuildView** window to view the output from the last build operation performed on an object.

8.0 Chapter 8. Using TeamConnection to Package Products

After you build an application to your satisfaction, you can distribute it to users. In this chapter we describe how to use TeamConnection to automate the packaging and distribution of your applications. The chapter is written for the person in your organization who is responsible for software distribution.

TeamConnection provides the following packaging support:

- Two electronic software distribution tools (see Figure 100):
 - **Gather/2**, which moves an application's parts into a single directory in preparation for distribution
 - **NVBridge/2**, a bridge tool that automates the installation and distribution of software or data using IBM NetView Distribution Manager/2 (NetView DM/2) as the distribution vehicle
- Two sample build scripts for connecting the Gather/2 and NVBridge/2 tools with TeamConnection user-defined builders
- A set of mini-utilities that can be used to develop customized electronic software distribution solutions

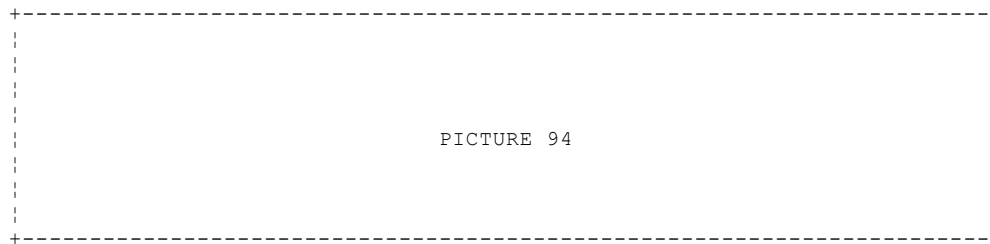


Figure 100. Electronic Distribution Using Gather/2 and NVBridge/2

To use TeamConnection in packaging a product, you can do any of the following tasks:

- Set up your application's build tree for packaging
- Use the *teamcpak gather* command
- Write a package file for the Gather/2 tool
- Use the *teamcpak nvbridge* command
- Write a package file for the NVBridge/2 tool

Subtopics

- 8.1 Setting Up Your Build Tree for Packaging
- 8.2 Using the Gather/2 Tool
- 8.3 Using the NVBridge/2 Tool

8.1 Setting Up Your Build Tree for Packaging

When TeamConnection builds an application, the application's build tree identifies the parts to be built and the tools to use in building it. Similarly, when you use TeamConnection for packaging an application, the build tree can define the parts to be packaged and the packaging tools. The output of a packaging step might be any of the following:

- ☐ The application's parts gathered into a new directory structure
- ☐ Distribution of the application through NVBridge/2
- ☐ Distribution of the application through some other distribution tool

In the sections that follow we talk about how to:

- ☐ Set up a build tree for the Gather/2 tool
- ☐ Set up a build tree for the NVBridge/2 tool
- ☐ Set up a build tree for other distribution tools.

Subtopics

- 8.1.1 Set Up a Build Tree for the Gather/2 Tool
- 8.1.2 Set Up a Build Tree for the NVBridge/2 Tool
- 8.1.3 Set Up a Build Tree for Other Distribution Tools

8.1.1 Set Up a Build Tree for the Gather/2 Tool

To gather the parts of your application into a single directory for distribution, you create an output part whose builder calls the Gather/2 tool, and you make this output part the top level of the build tree. For example, for the serata games application, SerataApplication, the build tree might look in part like Figure 101.

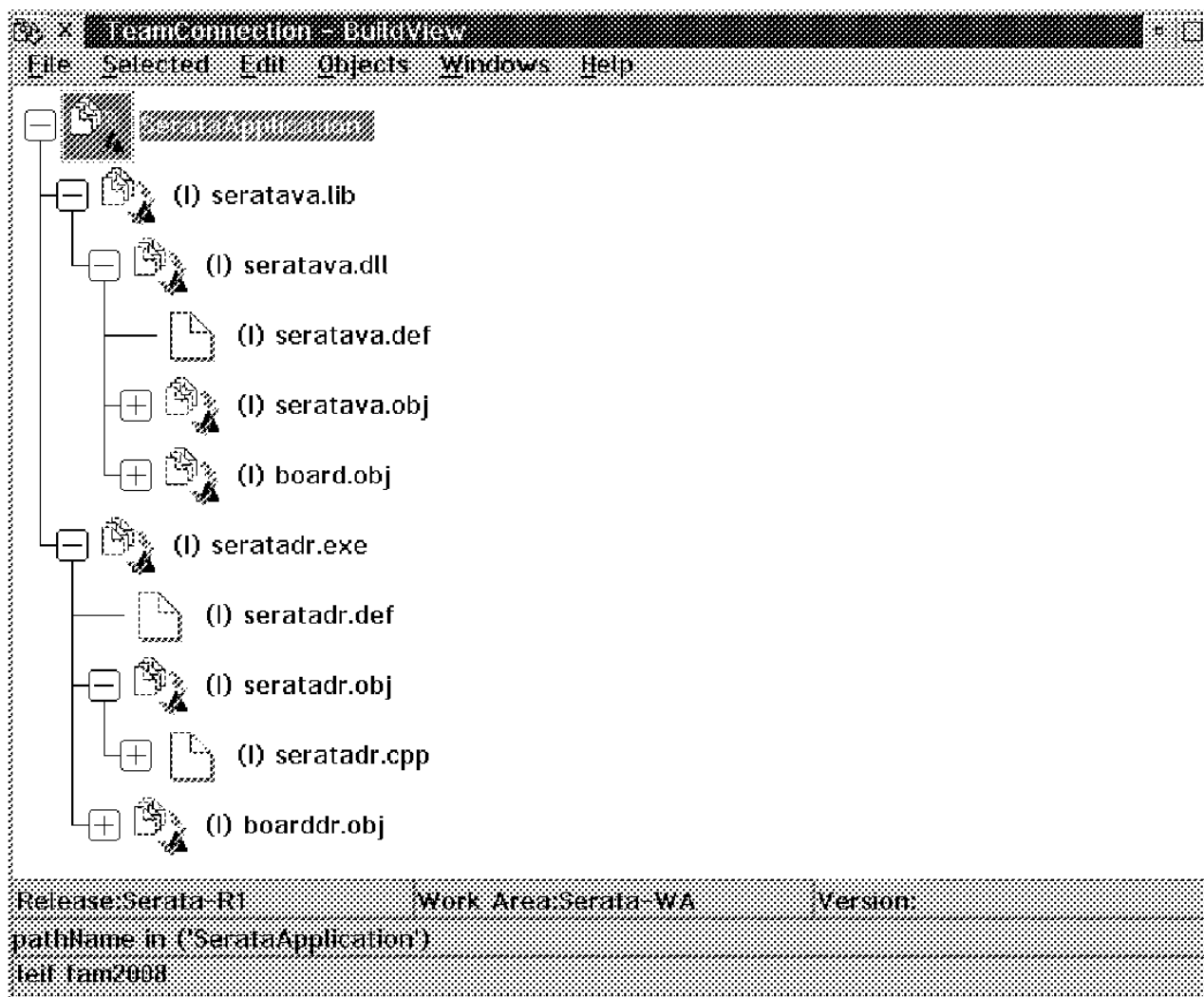


Figure 101. Part of the Build Tree for SerataApplication

After the SerataApplication is built, we have to get it to the test team. We could extract the application, but a simple extract would preserve the existing structure, with parts contained in directories according to their application component. A better structure might be one where all of the .dll files are in one directory, all of the .exe files in another, and so on. To move the parts into this structure, we do a different kind of build, using the Gather/2 tool:

1. We create the top-level part for the new build tree. The name of the part is *serata.gat*.
2. We write a package file, *serata.pkf*, that contains instructions for the Gather/2 tool and create this file as a TeamConnection part.
3. We create a builder, *Gatherer*, that calls the Gather/2 tool:

```
teamc Builder -create Gatherer -release Serata-R1 -script gather.cmd
              -environment os2 -condition "==" -value 0 -text
              -from D:\TEAMC\BIN\gather.cmd -parameters " -d -t"
              -timeout 10
```

The *gather.cmd* is a sample build script that is shipped with TeamConnection. It uses the *teamcpak* gather command.

4. We connect all of the parts that we want to gather as input to *serata.gat*.

As a result of our work, the build tree for *serata.gat* looks like Figure 102.

Figure 102. Adding the Gather/2 Step to the Build Tree

The package file, *serata.pkf*, specifies the directories into which the SerataApplication files are gathered, with *\seratgat* as the target root directory. When we build *serata.gat*, the *.dll* files are placed in *\seratgat\DLL*, the *.exe*, *.obj*, and *.lib* files are placed in *\seratgat\BIN*, and all the source files are placed in *\seratgat\SRC*. Instead of extracting the built application from TeamConnection, our test team can now pull the application from *\seratgat*.

If we want to gather the same files into a different target directory, all we have to do is write a different package file and connect the parts to a different parent.

8.1.2 Set Up a Build Tree for the NVBridge/2 Tool

If you have NetView DM/2 on your LAN, you can use the NVBridge/2 tool to distribute your application to users. Creating a build tree for the NVBridge/2 tool is similar to creating one for the Gather/2 tool. For example:

1. We create the top-level part for the new build tree. In this example, *serata.nvb* is the output file from the NVBridge/2 build. We use the following command:

```
teamc part -create serata.nvb -none -builder Nvbridge
          -release Serata-R1 -workarea Serata-WA1 -family fam2008
```

2. We write a package file, *seratnvb.pkf*, that contains instructions for the NVBridge/2 tool and create this file as a TeamConnection part:

```
teamc part -create seratnvb.pkf -text -parent serata.nvb
          -release Serata-R1 -workarea Serata-WA1 -family fam2008
```

For more about creating this package file, see "Writing a Package File" in topic 8.3.2.

3. We create a builder, *Nvbridge*, that calls the NVBridge/2 tool:

```
teamc builder -create Nvbridge -script nvbridge.cmd -release Serata-R1
             -environment os2 -condition == -value 0 -family fam2008
```

The build script for this builder uses the *teamcpak* command and its parameters.

4. We connect *serata.gat* and *seratnvb.pkf* to indicate that they are input to *serata.nvb*:

```
teamc part -connect serata.gat seratnvb.pkf -parent serata.nvb
          -family fam2008 -release Serata-R1 -workarea Serata-WA1
```

The build tree for *serata.nvb* looks like Figure 103.

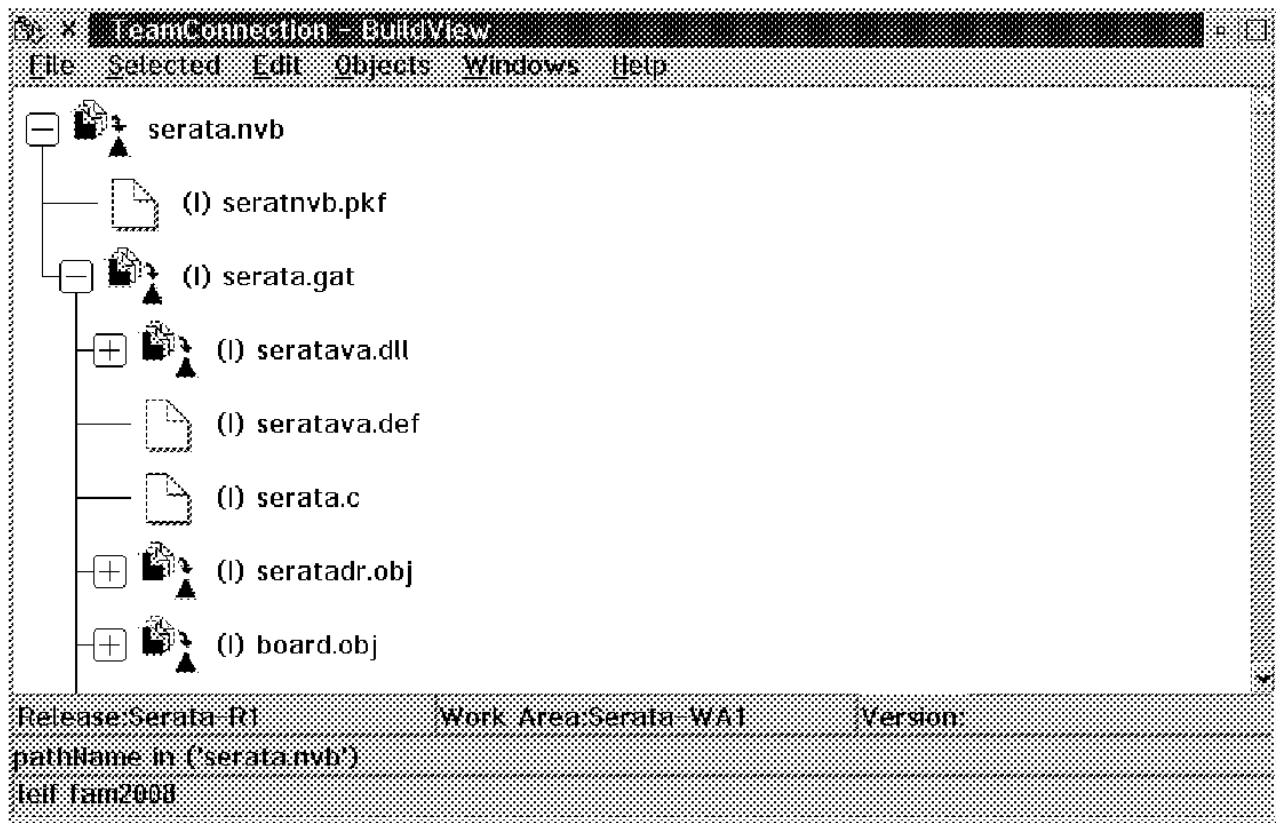


Figure 103. Adding the NVBridge/2 Step to the Build Tree

When we build *serata.nvb*, NVBridge/2 uses the rules in the package file to issue NetView DM/2 commands. Thus the entire package is distributed through NetView DM/2. Even though the gather step is not necessary when you want to distribute a package, it is very useful in preparing the package for distribution.

8.1.3 Set Up a Build Tree for Other Distribution Tools

The process of setting up a build tree for other distribution tools is similar to setting up a build tree for NVBridge/2. You create a top-level part in the build tree and a builder that invokes the distribution tool.

8.2 Using the Gather/2 Tool

The Gather/2 tool automates the movement of software and data from one directory to another on the same machine to prepare a package for electronic distribution. It can copy or erase files; it can create or delete directories.

The Gather/2 tool takes a list of input files and moves them into a directory structure as directed by a package specification file, also called a *package file*. You specify the target root directory path in the package file, along with a collection of rules that instruct which files to copy to which directories. How these files and directories are actually handled is controlled through option flags.

By writing different package specification files, you can take the same input files and transfer them to different target directory structures.

Figure 104 shows how the Gather/2 tool collects all files according to a user-created package file and moves them into a directory structure suitable for distribution.

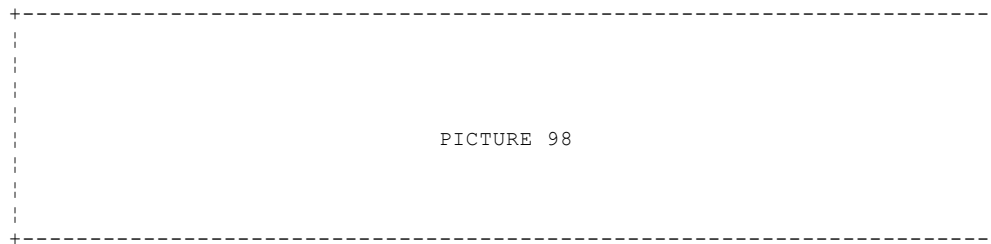


Figure 104. Using Gather/2

Subtopics

8.2.1 Using the teamcpak Command

8.2.2 Writing a Package File

8.2.1 Using the teamcpak Command

To start the Gather/2 tool, use the **teamcpak** command. This command is found in the directory where the TeamConnection family server is installed. If it is started from a build script, it does not have to be in the execution path of the machine from which the build is started.

The complete command syntax for *teamcpak gather* looks like this:

```
teamcpak [-i] [-o "String"] gather Input_file...
```

where:

-i

Specifies that only one *Input_file* is specified in the command: an include file containing the list of input files. This parameter is optional. If you specify **-i**, it must precede the *gather* flag.

-o "String"

Specifies that the string in quotes be passed to the Gather/2 tool. The opening quote must be followed by a blank. For a list of possible flags to be passed, see "Command Line Flags" in topic 8.2.1.1.

This parameter is optional. If you do not specify **-o**, the default settings for the tool are used.

If you specify **-o**, it must precede the *gather* flag.

gather

Specifies the tool to be invoked. If you specify **-i** or **-o**, it must precede this value.

Input_file...

Specifies the files to be copied and the name of the package specification file. You can specify this parameter in these ways:

- Specify the name of an include file, whose contents is a list of input files. One of these input files must be a package specification file with the extension *.pkf*. In this case, you must also specify the **-i** parameter.
- Specify a list of two or more files. One of the files must be a package specification file with the extension *.pkf*.
- Specify the directory from which the files are to be copied and the name of the package specification file.

If more than one package file is listed, the first package file on the command line or in the include file is used, and the others are treated as ordinary files.

Note: You must supply a value for the words that start with a capital letter, such as *String*. You must specify the command parameters in the order shown.

Subtopics

8.2.1.1 Command Line Flags

8.2.1.2 Examples of the teamcpak Gather Command

8.2.1.1 Command Line Flags

In the *teamcpak* command, you can specify the flags listed below, using the *-o* parameter. All of these flags are optional. If you do not specify a flag, the *teamcpak* command runs using defaults.

-a

Assume that the target tree structure might not exist. If a required directory does not exist, create it and continue processing.

This flag cannot be specified if the *-t* flag is specified.

If neither *-a* nor *-t* is specified, the default is to assume that the desired tree structure already exists. No verification is performed to confirm that the directories exist. If they do not, the condition is detected while the package file rules are being processed. If you stop the *teamcpak* command, some target directories might contain updated files.

-t

Ensure that the target tree is exactly the tree specified in the package file. If a directory of the same name exists, the Gather/2 tool does the following:

- ☐ Erases the entire contents of the directory and all of its subdirectories
- ☐ Destroys the directory and all of its subdirectories
- ☐ Performs an *mkdir* command to create the entire tree structure again as specified in the package file

This flag cannot be specified if the *-a* flag is specified.

If an *rmdir* command fails during processing, the *teamcpak* command stops.

If neither *-a* nor *-t* is specified, the default is to assume that the desired tree structure already exists. No verification is performed to confirm that the directories exist. If they do not, the condition is detected while the package file rules are being processed. If you stop the *teamcpak* command, some target directories might contain updated files.

-m

Accept missing source files.

If this flag is not specified, the default is to ensure that at least one file matches each source specification in the package file. If a match is not found, the Gather/2 tool stops processing.

-d

Accept duplicate files. If a file is found on the target directory that matches the source file specification, it is overwritten by the source file.

If this flag is not specified, the default is to ensure that no files on the target match the source file specification. For example, if the source specification is *s*.c*, and *serata.c* is found on the target, the Gather/2 tool stops processing.

-c

Clean up the target directories. Erase all files on all target directories that existed before writing source files to those directories. No confirmation messages are issued, and permission errors are ignored.

If this flag is not specified, the default is to write the source files into the target directories without erasing existing files.

-e

End with delete. This action removes all source files and directories after the Gather/2 tool successfully completes.

If this flag is not specified, the default is to end without deleting source files and directories.

-x

Abort without recovery. If the program does not end successfully, no attempt is made to restore the file system.

If this flag is not specified, the Gather/2 tool attempts to restore the file system if the program does not end successfully. To do this, the tool first backs up the file system. The backup directory is the value of the TMP environment variable.

8.2.1.2 Examples of the teamcpak Gather Command

The following are examples of the *teamcpak gather* command:

```
teamcpak gather d:\Serata serata.pkf
teamcpak gather serata.exe board.exe seratadr.exe serata.pkf
```

In the first example, an input source directory is specified. In the second example, a list of files is specified. In both cases, the files are to be copied into target directories as specified in the *serata.pkf* file.

```
teamcpak -i -o " -t -m -x" gather myfiles.lst
```

The *myfiles.lst* file contains a list of files to be transformed by the Gather/2 tool, and the name of the package file to be used in the gather. The *-o "-t -m -x"* parameter passes three flags to the tool:

- ☐ **-t** specifies that, if the target directories already exist, they should be destroyed and re-created.
- ☐ **-m** specifies that processing continues even if a source file cannot be found.
- ☐ **-x** specifies that, if the program does not end successfully, the file system is left as is, with no attempt to restore it.

8.2.2 Writing a Package File

Use the package file to specify the target directories and the rules for copying files for a gather operation. You can also specify user exit programs to run before, during, or after the gather operation. A sample package file named *gather.pkf* is shipped with TeamConnection. You can customize it for your own gather operations.

Subtopics

8.2.2.1 Syntax Rules

8.2.2.1 Syntax Rules

Follow these syntax rules when you write a package file:

- Package files are free format. Text is not positional, and many statements can exist on the same line.
- Comments can appear anywhere within the file. Use the `#|` and `|#` characters as delimiters, as shown in the following example:


```
#| This is a comment |#
```
- Package file keywords must be prefixed with a left parenthesis and have a corresponding balanced right parenthesis to end the scope.
- If the value for a keyword is a string that contains blanks or parentheses, enclose the string in double quotes.

The syntax of a package file for the Gather/2 tool is given below. Keywords must appear in the order shown. The first letter of an argument is capitalized; you must supply these values.

```
(DATA
  (PACKAGEFORMAT gather)
  (TARGETROOT Target_root_path)
  (RULE
    (SOURCE Filename...)
    (TARGET Target_path)
    [(EXITPRIOR String... | EXITREPLACE String... | EXITPOST String...)])
  )
  .
  .
  [(EXITPRIOR String...)]
  [(EXITPOST String...)]
)
```

where:

DATA

This keyword is required. It must be the first keyword in the package file, and it can be specified only once. All other keywords are nested within the *DATA* clause.

PACKAGEFORMAT *gather*

This keyword is required. It can be specified only once. It tells the *teamcpak* command that this package file is for Gather/2.

TARGETROOT *Target_root_path*

This keyword is required. It can be specified only once.

Use this keyword to identify the target root directory. Source files are copied to this directory as specified by the *RULE* statements.

Follow these guidelines when you select your *TARGETROOT* values:

- Include the drive letter along with the target directory.
- Specify a directory that contains few if any subdirectories that are unrelated to the data you are moving.
- If you specify a drive's root directory (drive:\), run the *teamcpak* command, using the defaults or only the `-x` or `-x -a` flags.
- Do not set the value of *TARGETROOT* to `drive:\` under the following circumstances:
 - The *TARGETROOT* drive is the same as the drive from which the *teamcpak* command is run, and you have recovery set (that is, you have not specified `-o "-x"`).
 - The logical drive for the *TARGETROOT* has less than 50% free space, and you have recovery set (that is, you have not specified `-o "-x"`).

RULE

This keyword is required. You can use one or more *RULE* keywords within a Gather/2 package file.

Each *RULE* clause represents a set of Gather/2 operations targeted for one target subdirectory. A *RULE* clause must contain one *SOURCE* and one *TARGET* keyword. The files in the *SOURCE* directory are copied to the *TARGET* path. The target path is derived by concatenating the value of *TARGETROOT* with a backslash (\), followed by the value of the *TARGET* keyword specified in the *RULE* clause.

A *RULE* clause can also contain one user exit clause: *EXITPRIOR*, *EXITPOST*, or *EXITREPLACE*.

The following example copies all **.exe*, **.cmd*, and **.hlp* files to target directory *e:\serata\bin*.

```
(DATA
.
.
(TARGETROOT  e:\serata )
.
.
(RULE
  (SOURCE  *.exe  *.cmd  *.hlp)
  (TARGET  bin   )
)
.
.
)
```

SOURCE <list of file specifications>

This keyword is required once for each *RULE* clause. It must be the first keyword within the *RULE* clause.

This keyword specifies the files to be copied to the path specified by the *TARGET* keyword. Specify a list of file specifications separated by blanks. You can use the wildcard characters supported by the OS/2 command line.

The directory from which these files are copied depends on how the input files are specified in the *teamcpak* command:

- If the *teamcpak* command specifies a source directory, the files specified in the *SOURCE* keyword come from that directory or subdirectories of it. The full path of the source files is constructed by concatenating the directory from the *teamcpak* command with a backslash (\), followed by the file specifications found in the *SOURCE* keyword. You can specify subdirectories in the *SOURCE* file specifications.
- If the *teamcpak* command specifies a list of files, these files are first copied to a temporary directory, then copied from there to the *TARGET* directories. In this case, you can use OS/2 wildcards to specify multiple file names in the *SOURCE* file specifications, but you cannot specify subdirectories.

In the following example, directory *d:\serata* is specified on the *teamcpak* command:

```
teamcpak -o "-x -t -m" gather d:\serata serata.pkf
```

The resulting source path is the concatenation of *d:\serata* with the *SOURCE* file specifications. Therefore, all of the *.exe* files in directory *d:\serata\bin* are copied to target directory *e:\demoapp\bin*.

```
(DATA
  (TARGETROOT  e:\serata)
  .
  .
  (RULE
    (SOURCE  bin\*.exe)
    (TARGET  bin)
  )
  .
  .
)
```

In the following example, a list of input files is specified on the *teamcpak* command:

```
teamcpak -o "-x -m" gather c:\a.exe c:\b.exe d:\rexx\*.cmd serata.pkf
```

The resulting source path for the files in the *SOURCE* clause is the concatenation of the *teamcpak* temporary directory with the *SOURCE* file specifications. Therefore, the source for the *.exe* files is *d:\teamcpak.@@@*.exe*. Input files *d:\teamcpak.@@@a.exe* and *d:\teamcpak.@@@b.exe* are copied to directory *e:\serata*.

```
(DATA
  (TARGETROOT  e:\serata)
  .
  .
  (RULE
    (SOURCE  *.exe   )
    (TARGET  targetroot)
  )
  .
  .
)
```

TARGET *Target_path*

This keyword is required one time in each *RULE* clause. It must follow the *SOURCE* keyword.

The value specified by this keyword is used to construct the target path into which the files specified by the *SOURCE* keyword are copied.

The value of the *TARGETROOT* keyword is concatenated with a backslash (\), followed by the value of the *TARGET* keyword.

If you specify *targetroot* as the value, files are copied directly to the target root directory, not to a subdirectory.

In the first *RULE* clause of this example, files are copied to the target directory *f:\serata\bin\files*. In the second *RULE* clause, the target directory is *f:\serata*.

```
(DATA
  (TARGETROOT  f:\serata )
  .
  .
  (RULE
    (SOURCE  *.bin *.dll      )
    (TARGET  bin\files )
  )
  (RULE
    (SOURCE  *.hlp           )
    (TARGET  targetroot )
    (TARGET  targetroot )
  )
  .
  .
)
```

EXITPRIOR, EXITREPLACE, and EXITPOST String...

These keywords are optional. They specify a user exit program to run as part of the Gather/2 operation.

To specify an exit that is global to the Gather/2 operation, specify *EXITPRIOR* or *EXITPOST* in the *DATA* clause. You can specify each of these keywords only once in the *DATA* clause. These keywords must come after all of the *RULE* clauses. *EXITREPLACE* cannot be used in the *DATA* clause.

You can also specify an exit that is specific to one *RULE* clause. Only one exit keyword is allowed in each *RULE* clause. These keywords accept a list of strings separated by spaces. The first string is the name of the program to execute. The strings that follow are its parameters.

Using Exit Keywords in the DATA Clause: When used within a *DATA* clause, the exit keywords identify a program or command to be executed within a command shell. *EXITPRIOR* executes before all *RULE* statements have been processed; *EXITPOST*, after all *RULE* statements.

The exit keywords accept any executable file or command. The exit program must return an integer return value, with zero meaning the exit was successful.

Using Exit Keywords in the RULE Clause: *EXITPRIOR*, *EXITREPLACE*, and *EXITPOST* are optional within a *RULE* clause. Only one can be specified in any given *RULE* clause.

When used within a *RULE* clause, the exit keywords identify a program or command to be executed within a command shell before, after, or in place of processing of each Gather/2 copy operation. The exit program is called once for each *SOURCE* specification entry within the *SOURCE* clause.

Parameters are separated by spaces and passed to the exit in this order:

- ☐ Any parameters included in the invocation string
- ☐ The resolved *SOURCE* file specifications
- ☐ The resolved *TARGET* specification

The exit keyword accepts any executable file or command. The exit program must return an integer return value, zero meaning successful; it must also accept or ignore the additional Gather/2 parameters added to the end of the invocation string. When used in the context of the *RULE* clause, exit keywords must follow the *TARGET* keyword.

Using Exit Keywords--An Example: In the following example, the first *EXITPRIOR* statement relates to the *DATA* clause and specifies a user backup

exit program that executes before performing Gather/2 copy operations. This backup exit is passed two flags. The command stream executed in an OS/2 shell is:

```
"e:\util\backup.cmd \i \t"
```

The second occurrence of the keyword illustrates how to use it in the context of a *RULE* clause. In this example, an encryption program will run against each source file specification. The exit program is passed the \k:347867 key option, the value for the source specification, and the value for the target specification. In this example, the command stream executed in an OS/2 shell is:

```
"encrypt \k:347867 d:\demoapp\*.exe f:\demoapp\bin":
```

The package file looks like this:

```
(DATA
  (PACKAGEFORMAT gather)
  (TARGETROOT d:\tcws)
  (RULE
    (SOURCE *.exe *.cmd)
    (TARGET exe)
    #|this program will be run for each source file|#
    (EXITPRIOR encrypt \k:347867 )
  )
  (EXITPRIOR "e:\util\backup.cmd \i \t" )
)
```

8.3 Using the NVBridge/2 Tool

The NVBridge tool supports automated distribution between a single NetView DM/2 change control (CC) server and its LAN-connected CC clients. It also supports remote distribution to APPC-connected NetView DM/2 servers and mainstream servers.

A sample build script named *nvbridge.cmd* is shipped with TeamConnection. It can be invoked within a TeamConnection builder. This build script maps TeamConnection build parameters to the command line syntax for invoking the NVBridge/2 tool through the *teamcpak* command line interface.

You can use NVBridge/2 as a builder for packaging in two ways:

- Integrate it with the gather step, so that the Gather/2 tool leaves the package files in a directory from which NVBridge picks them up.
- Use it without the gather step. In this case, the build script for NVBridge/2 must set up the directory and move files into it to interface correctly with the *teamcpak* command.

For information about setting up a build tree for running NVBridge/2, see "Set Up a Build Tree for the NVBridge/2 Tool" in topic 8.1.2.

NVBridge produces the following NetView DM/2 output files:

A change file

This file, containing all of the software deliverables, is stored in the *fsdata* subdirectory of the NetView DM/2 directory. The file name is *buildID.cf*, where *buildID* is the ID specified in the *-o "-b"* flag of the *teamcpak* command.

A procedure file

This file is stored in the *fsdata* subdirectory of the NetView DM/2 directory. It contains the command instructions for uninstalling an installed software object during its next build. The file name is system-generated.

A flat data file

This file, containing the text data of mail information that accompanies other objects, is stored in the *fsdata* subdirectory of the NetView DM/2 directory. The file name is system-generated.

Catalog entries for the generated change file, procedure file, and mail notification object

These files are named according to the following naming convention:

_Tx_corporation_ID.buildID.xxx.0.0

where:

- *x* is an identifier for the TeamConnection server. The default is *C*.
- *corporation_ID* is a string of up to 10 characters. The default is *NULLCORP*.
- *buildID* is a string of up to 16 characters, representing the ID specified on the *-b* flag of the *teamcpak nvbridge* command.
- *xxx* identifies the file. The following values are used:
 - *REF* for the generated change file
 - *CMD* for the generated uninstall procedure
 - *MAIL* for the generated mail notification object

Figure 105 shows the software distribution model for NVBridge/2. The software distribution model is based on *complete builds* and hence works from the assumption that each software delivery through NVBridge/2 is a new delivery.

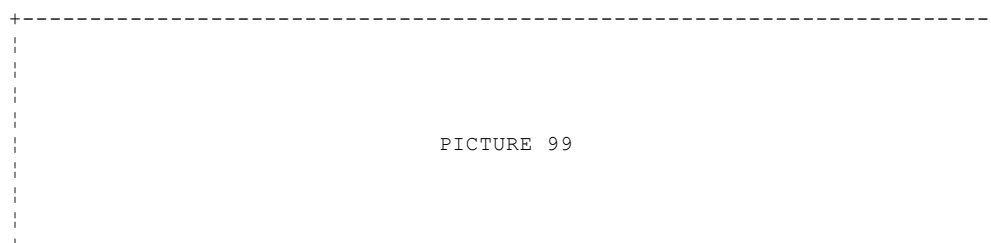


Figure 105. The NVBridge/2 Software Distribution Model

Subtopics

8.3.1 Using the teamcpak Command

8.3.2 Writing a Package File
8.3.3 Problem Determination
8.3.4 NVBridge Utilities

8.3.1 Using the teamcpak Command

To start the NVBridge tool, use the *teamcpak* command. This command is found in the directory where the TeamConnection family server is installed. If it is started from a build script, it does not have to be in the execution path of the machine from which the build is started.

The complete syntax for the *teamcpak nvbridge* command looks like this:

```
teamcpak -o "string" nvbridge input_source-directory package_specification_file
```

where:

-o "string"

Specifies that the string in quotes be passed to the NVBridge/2 tool. For a list of possible flags to be passed, see "Command Line Flags" in topic 8.3.1.1.

Input_source_directory

A directory containing all of the files and subdirectories of the software to be distributed using NetView DM/2. You must specify this directory as an absolute path with no wildcard characters.

The source root directory can be created in an earlier build step. The Gather/2 tool can be used to create the source root directory and move the files into it as the *TARGETROOT* directory.

All files in the source root directory are included in the NetView DM/2 object that is built and distributed through NVBridge/2. It is important that the source root directory contain only the subdirectories and files required for the software package distribution.

File names in the source directory cannot contain blanks. Only nonhidden files are supported. The names of the files are reproduced on the target NetView DM/2 CC clients. Therefore, HPFS file names can be included only if the target NetView DM/2 CC clients listed in the package file have target drives that support HPFS.

If you are using the uninstall function, the uninstall program must reside in the source directory. If you use the *MAIL* keyword for remote servers, the *MAIL* text file that is sent to the remote servers must also reside within this directory.

package_specification_file

A file describing how the NVBridge/2 function is to build, catalog, and distribute software. Additional controls of NVBridge/2 processing are provided through the optional command line flags described in "Command Line Flags" in topic 8.3.1.1.

Note: You must specify the command parameters in the order shown.

Subtopics

8.3.1.1 Command Line Flags

8.3.1.2 Examples of the teamcpak nvbridge Command

8.3.1.1 Command Line Flags

You can specify the following flags in the *teamcpak* command, using the *-o* parameter. All of these flags except *-b* are optional.

-b:buildID

This flag is required. The build ID represents the software to be distributed. It is a string of up to 16 alphabetic characters or underscores. It cannot contain blanks or other special characters. If you are distributing software to NetView DM/2 clients whose target drives do not support HPFS, the build ID can be only eight characters long.

-v

Use this flag to verify that NetView DM/2 CC clients listed in the *INSTALLS* keyword of the package file are defined to the NetView server and are in an active state. Verification takes place one time at the start of the *teamcpak* command. Changes that occur after verification are not detected.

Verification fails if one or more clients is in running rather than active status. Running status indicates that the client currently has a NetView command in progress.

If this flag is not specified, NVBridge/2 ignores errors resulting from undefined or inactive clients. If the *SENDS* keyword is specified in the package file, this flag also verifies the following:

- ☐ All of the remote destinations are defined in the NetView DM/2 remote destinations table.
- ☐ Any transmission queues associated with the remote destinations are in a released state.
- ☐ The transmission queues are empty.

If the *INSTALLS* or *SENDS* keyword is not specified in the package file, verification always succeeds.

-m

Use this flag to monitor NVBridge/2-generated requests to NetView DM/2. If any install, uninstall, or send requests do not complete successfully, messages are generated.

This flag works with the *CLIENTINTERVAL*, *SENDINTERVAL*, and *ATTEMPTS* package file keywords. See "Writing a Package File" in topic 8.3.2 for details about using them to control monitoring durations and limits.

If you specify this flag, NVBridge/2 continues to run as long as it takes to install on all the designated clients and send to remote destinations, or until the durations set by the package file keywords are exceeded.

If this flag is not specified, NVBridge/2 submits the install and send requests and then ends without waiting for them to complete. In this case, you can use NetView DM/2 functions to track these pending requests.

-r

Use this flag to retry an earlier failed attempt to install. Using the same package file, you can correct install failures without rebuilding the software object and without reinstalling on clients that were successful the previous time.

The *TEST* and *SENDS* keywords in the package file are ignored, because testing and sending are assumed to already have taken place.

-a

Use this flag to issue NetView *ACTIVATE* requests. Such requests cause the client machine to reboot after installation processing and *SEND* requests are complete. This is the last request that NVBridge/2 performs before completion.

This flag is ignored if the *TEST* and *INSTALLS* keywords are not specified in the package file. If you specify this flag, you must also specify the *-m* flag.

-l

Use this flag to install software in a NetView DM/2 service area. This allows installation of files that replace system-locked files.

A NetView *ACTIVATE* request is required to move the files from the temporary service area and have updates to the *CONFIG.SYS* file take effect. You can use the *-a* flag to request the *ACTIVATE*. If you do not use the *-a* flag, you must use NetView to reboot the clients

manually.

If you use this flag, the installation program must be installed on the client or reside in the CC server's NetView DM/2 shared areas. This flag is ignored if the *TEST* and *INSTALLS* keywords are not specified in the package file.

-t: *time*

Use this flag to set a timer so that NVBridge/2 runs at a later time. For example, you can invoke NVBridge/2, go home, and let it start during the night. This timer applies to all NVBridge/2 functions, not individual *INSTALLS* or *SENDS* requests.

Valid values for time are 0000 to 2359. For example, specify **-t:0000** to run NVBridge/2 starting at midnight. Specify **-t:1200** to start NVBridge/2 at noon.

-f

Use this flag to prevent the sending of objects that have install or uninstall problems. If any *INSTALL* or *UNINSTALL* requests fail, NVBridge/2 returns a nonzero return code and purges any pending requests.

If this flag is not specified, NVBridge/2 returns a return code of 0 and continues with *SENDS* requests, even if some *UNINSTALL* or *INSTALL* requests fail to complete.

8.3.1.2 Examples of the teamcpak nvbridge Command

The following is an example of the *teamcpak nvbridge* command:

```
teamcpak -o "-b:demoapp -m -v" nvbridge d:\demoapp demoapp.pkf
```

The input source directory, *d:\demoapp*, contains the files to be installed. The *demoapp.pkf* file is the package specification file.

The *-o* parameter passes three flags to the NVBridge/2 tool:

- ☐ **-b:demoapp** specifies that *demoapp* is the build ID.
- ☐ **-m** specifies that NVBridge/2 monitor requests to NetView DM/2 to completion.
- ☐ **-v** specifies that NVBridge/2 verify that all clients are defined to NetView DM/2 and active.

8.3.2 *Writing a Package File*

This section describes the NetView DM/2 package file keywords and their effect on normal processing behavior. A sample package file named *nvbridge.pkf* is shipped with TeamConnection. You can customize it for your own use.

Subtopics

8.3.2.1 Syntax Rules

8.3.2.2 Keywords for an NVBridge/2 Package File

8.3.2.1 Syntax Rules

Follow these syntax rules when you write a package file:

- Package files are free format. Text is not positional, and many statements can exist on the same line.
- Comments can appear anywhere within the file. Use the `#|` and `|#` characters as delimiters, as shown in the following example:


```
#| This is a comment |#
```
- Package file keywords must be prefixed with a left parenthesis and must have a corresponding balanced right parenthesis to end the scope of the keyword.
- If the value for a keyword is a string that contains blanks or parentheses, enclose the string in double quotes.

The syntax of a package file for NVBridge/2 is given below. The order of the keywords in the *NVGLOALS* clause does not matter; all other keywords must appear in the order shown. You must supply the values for the strings that are shown in italics.

```
(DATA
(PACKAGEFORMAT nvbridge
(NVGLOALS
  [(MAIL filename)]
  [(TEAMCSERV x)]
  [(CORPID name)]
  (INSTALLDIR path)
  [(INSTALLPGM path\filename)]
  [(IPARMS parameters)]
  [(UNINSTALLPGM filename)]
  [(CLIENTINTERVAL n)]
  [(SENDINTERVAL n)]
  [(ATTEMPTS n)]
)
[(TEST
  (ENTRY client)
)]
[(INSTALLS
  (ENTRY client target_directory)
  .
  .
)]
[(SENDS
  destination
  .
  .
)]
)
```

8.3.2.2 Keywords for an NVBridge/2 Package File

This section explains the NVBridge/2 package file keywords.

DATA

This keyword is required. It must be the first keyword in the package file, and it can be specified only once. All other keywords are nested within the DATA clause.

Example:

```
(DATA
.
.
  other keywords go here
.
.)
```

PACKAGEFORMAT *nvbridge*

This required keyword must be the first keyword within the DATA clause. It can be specified only once. It tells the *teamcpak* command that this package file is for NVBridge/2.

Example:

```
(DATA
.
.
  (PACKAGEFORMAT nvbridge)
.
.)
```

NVGLOBALS

This required keyword must follow the *PACKAGEFORMAT* keyword within the DATA clause. It can be specified only once. All of the global and default keywords for the package file are specified in the *NVGLOBALS* clause. At least one *INSTALLDIR* keyword must be specified within this clause; all other *NVGLOBALS* keywords are optional.

Keywords in the *NVGLOBALS* clause can appear in any order.

Example:

```
(DATA
.
.
  (NVGLOBALS
.
    NVGLOBALS keywords go here
.
  )
.
.)
```

MAIL *filename*

This optional keyword can appear at any place within the *NVGLOBALS* clause. It can be specified only one time.

This keyword identifies a text file that is sent as a mail notification object to all destinations listed in the *SENDS* keyword. The file name specified must contain only the file name and extension, no path information. NVBridge/2 searches the input directory specified on the command and uses the first file that matches this file name.

The mail object is created and cataloged on the local server along with the *.ref* and *.cmd* objects. Mail objects typically take the form of readme files.

Example:

In the following example, the input directory is searched for the first occurrence of the *readme.txt* file during NetView DM/2 processing. This file is used to create a new *.mail* object cataloged to NetView DM/2. This object is then sent along with the *.ref* and *.cmd* objects during the processing of any *SENDS* requests.

```
(DATA
.
.
(NVGLOBALS
.
(MAIL  readme.txt )
.
)
.
.
)
```

TEAMCSERV *x*

This optional keyword can appear at any place within the *NVGLOBALS* clause. It can be specified only once.

This keyword specifies a one-character alphanumeric ID to be used in global name generation. It identifies the TeamConnection family server that built and distributed the global named object. If this keyword is specified, the first four characters of the global name are *_Tx_*. If this keyword is not specified, the first characters of the global name default to *_TC_*.

Example:

An example of a NetView DM/2 object name based on the following example is *_t1_nullcorp.serata.ref.0.0*.

```
(DATA
.
.
(NVGLOBALS
.
(TeamCSERV 1 )
.
)
.
.
)
```

CORPID *name*

This optional keyword can appear at any place within the *NVGLOBALS* clause. It can be specified only once.

This keyword is used in constructing the global names of NetView objects generated by NVBridge/2. You can use it to further identify the NetView objects, for example, by specifying the name of your company. The name specified in this keyword follows the value of the *TEAMCSERV* keyword in global names.

The name specified in this keyword can be from 1 to 10 alphanumeric characters. If this keyword is not specified, it defaults to the value *NULLCORP*.

Example:

An example of a NetView DM/2 object name based on the following example is *_t1_ibmcorp.serata.ref.0.0*.

```
(DATA
.
.
(NVGLOBALS
.
(CORPID  ibmcorp)
.
)
.
.
)
```

INSTALLDIR *path*

This required keyword can appear at any place within the *NVGLOBALS* clause. It can be specified only once. This keyword defines the

default workstation directory that is to be created as the root software directory on the clients.

The path must be an absolute file specification, including a valid drive. You can override this value for individual clients, within the *ENTRY* clause of the *INSTALLS* keyword. This target directory need not exist on the CC clients. NetView creates this directory and populates it with the same data as in the input directory specified on the *teamcpak* command.

Example:

In the following example, the *e:\serata* path is used as the target drive and directory for all CC clients listed in the *INSTALLS* keyword. NetView DM/2 will install the files and subdirectory structure specified on the *teamcpak* command within target directory *e:\serata* on each CC client machine.

```
(DATA
.
.
(NVGLOBS
.
(INSTALLDIR e:\serata )
.
)
.
.
)
```

INSTALLPGM *path\filename*

This optional keyword can appear at any place within the *NVGLOBS* clause. It can be specified only once. This keyword defines the installation program to be invoked after the software files are copied on the CC client machine. If this keyword is not specified, the installation process includes only the movement of files, and an installation program is not invoked.

The installation program can be specific to the software just installed. In this case, the installation program is itself a file that was copied. Alternatively, the installation program can be a separate software product installed previously on the client machine. The installation program must meet the following criteria:

- ❑ It must be in the client's execution path, or the executable name must include its fully qualified path.
- ❑ The return value of the program must be 0 if you want NetView DM/2 to catalog the installation as successful. If the program returns a value other than 0, NetView DM/2 concludes that the installation was a failure.
- ❑ The installation program cannot be interactive; that is, it cannot expect user input.

The installation program can invoke other programs.

The *INSTALLS* entries and hence the path to the installation program can differ from client to client. As a result, the installation program often needs to know the target directory into which the software was installed. To aid with NetView environmental information such as this, you can include the following macros anywhere and as many times as needed within the *INSTALLPGM* value:

\$(TargetDir) Substitutes the path of the directory where the software was installed on the client

\$(WorkStatName) Substitutes the name of the client

Note: When including these macros, enclose the value in quotes.

Example:

In the following example, after the software and data files are installed on the CC clients, an installation program, *inst.exe*, is invoked to update each client's *CONFIG.SYS* file. The path to this installation program is within the software being installed and is resolved to a fully qualified path of *e:\serata\bin\inst.exe* on each CC client. In this example, the value of *INSTALLPGM* is enclosed in quotes because of the macro within the string.

```
(DATA
.
.
(NVGLOBALS
.
(INSTALLDIR e:\serata )
.
(INSTALLPGM "$(TargetDir)\bin\inst.exe" )
.
)
.
.
)
```

IPARMS *parameters*

This optional keyword can appear at any place within the *NVGLOBALS* clause. It can be specified only once.

This keyword defines the input parameters passed to the installation program specified on the *INSTALLPGM* keyword. If this keyword is not specified, it defaults to a null string. This keyword is ignored if the *INSTALLPGM* keyword is not also specified.

The *INSTALLS* entries and hence the path to the installation program can differ from client to client. As a result, the installation program often has to know the target directory into which the software was installed. To aid with NetView environmental information such as this, you can include the following macros anywhere and as many times as needed within the *IPARMS* value:

\$(TargetDir) Substitutes the path of the directory where the software was installed on the client

\$(WorkStatName) Substitutes the name of the client

Note: When including these macros, enclose the value in quotes.

Example:

```
(DATA
.
.
(NVGLOBALS
.
(IPARMS "\i \t \g \d:$(TargetDir)" )
.
)
.
.
)
```

UNINSTALLPGM *filename*

This optional keyword can appear at any place within the *NVGLOBALS* clause. It can be specified only once. This keyword defines the program that will be used to uninstall this version of the software when its next version is installed. For example, if you are installing Version 1.0 of your package, use this keyword to specify the program that will be used to uninstall Version 1.0 when Version 1.1 is installed.

The uninstall program must be in the input software directory. NVBridge/2 searches the input directory specified on the *teamcpak* command and uses the first file that matches the file name specified on this keyword.

If both this keyword and the *INSTALLS* or *TEST* keyword are specified, a NetView *PROCEDURE* object is created along with the software object. When the next version of this package is installed, this uninstall procedure is run against previous install clients, before the removal of the old version of the software and the creation of the new NetView software object. NVBridge/2 tracks only whether the uninstall operations complete, not whether they are successful. The uninstall program can itself invoke other programs.

Example:

In the example below, the input directory is searched for the first occurrence of the *uninst.cmd* file during NetView DM/2 processing. This

file is used to create a new uninstall *PROCEDURE* object cataloged to NetView DM/2.

```
(DATA
.
.
(NVGLOBS
.
  (UNINSTALLPGM  uninst.cmd )
.
)
.
.
)
```

CLIENTINTERVAL *n*

This optional keyword can appear at any place within the *NVGLOBS* clause. It can be specified only once.

This keyword identifies the sleep interval used for monitoring client operations such as uninstalling and installing. It is used with the *-m* flag on the *teamcpak* command. Specify a value in seconds. Valid values are from 10 to 600. The default is 15 sec.

Example:

```
(DATA
.
.
(NVGLOBS
.
  (CLIENTINTERVAL  60      )
.
)
.
.
)
```

SENDINTERVAL *n*

This optional keyword can appear at any place within the *NVGLOBS* clause. It can be specified only once.

This keyword identifies the sleep interval used for monitoring *SEND* operations. It is used with the *-m* flag on the *teamcpak* command. Specify a value in seconds. Valid values are from 10 to 600. The default is 15 sec.

Example:

```
(DATA
.
.
(NVGLOBS
.
  (SENDINTERVAL  40      )
.
)
.
.
)
```

ATTEMPTS *n*

This optional keyword can appear at any place within the *NVGLOBS* clause. It can be specified only once.

This keyword identifies the maximum number of attempts to monitor NVBridge/2 operations. Specify a number from 1 to 6. The default is 4. This value is combined with the *CLIENTINTERVAL* and *SENDINTERVAL* values to compute the maximum monitoring time. For each operation per client or prior remote destination, NVBridge/2 uses the following formul

Monitor and check every x seconds up to y times

where x is the value for *CLIENTINTERVAL* or *SENDINTERVAL*, and y is the

value for *ATTEMPTS*.

Example:

If you are installing to four clients and you specify the *-m* flag on the *teamcpak* command, by default NVBridge/2 monitors every 15 seconds, up to four attempts. It repeats this four times for each of the four clients, resulting in a total of 16 attempts at 15-sec intervals.

```
(DATA
.
.
(NVGLOBALS
.
(ATTEMPTS 4 )
.
)
.
.
)
```

TEST

This optional keyword appears within the *DATA* clause before the *INSTALLS* or *SENDS* keyword. It can be specified only once.

This keyword identifies a single NetView DM/2 CC client to be used as a test machine. All normal processing is first performed against this single client. If everything succeeds, normal processing continues for all clients listed in the *INSTALLS* keyword; otherwise normal processing stops. This keyword accepts a single *ENTRY* keyword, which identifies the client. Do not repeat this client value in the *INSTALLS* entries, because this might cause NVBridge/2 to fail.

Example:

In the example below, the test CC client named *CLIENT1* will be used to perform a test installation. The software and data will be installed to target client directory *e:\serata*.

```
(DATA
.
.
(TEST
(ENTRY CLIENT1 e:\serata)
)
.
.
)
```

INSTALLS

This optional keyword is specified within the scope of the *DATA* clause. It must follow the *NVGLOBALS* and *TEST* keywords. It can be specified only one time.

This keyword identifies the list of *ENTRY* keywords specifying the clients where the software object is to be installed. Duplicate *ENTRY* keywords for the same client are ignored.

ENTRY *client, client target directory*

This required keyword is specified within the scope of the *TEST* or *INSTALLS* clause. It can be specified many times.

This keyword identifies a NetView DM/2-defined CC client workstation on which the software is to be installed. For each *ENTRY* keyword, you must specify the name of a CC client machine.

Optionally, you can also specify a target installation directory for the client. This directory overrides the directory specified in the *INSTALLDIR* keyword. The target directory must be an absolute file specification, including a valid drive. If this value is found to be invalid, NetView uses the default value found in the *INSTALLDIR* keyword.

Example:

In the example below, the .ref object created by NetView DM/2 will be installed to *CLIENT1*, *CLIENT2*, *CLIENT3*, and *CLIENT4* CC client machines. In the case of *CLIENT2* and *CLIENT4*, the software is installed in the

default *INSTALLDIR* value of *d:\serata*. For the others, the target directory in the *ENTRY* keyword overrides the *INSTALLDIR* value.

```
(DATA
.
(NVGLOBS
.
(INSTALLDIR d:\demoapp )
.
)
.
(INSTALLS
(ENTRY CLIENT1 e:\demoapp)
(ENTRY CLIENT2 )
(ENTRY CLIENT3 c:\demoapp)
(ENTRY CLIENT4 )
)
.
.
)
```

SENDS destination ...

This optional keyword is specified within the scope of the *DATA* clause. It must follow the *NVGLOBS*, *TEST*, and *INSTALLS* keywords. It can be specified only once.

This keyword identifies the list of remote destinations that are to receive NVBridge/2-created objects. These destinations are APPC-connected NetView DM family servers.

Each remote destination in this list should be configured to accept creation or replacement of cataloged objects. If the remote server does not allow incoming *SENDS* of objects, NVBridge/2 cannot send objects to it. Also, if the remote server accepts only creates and not replacements, NVBridge/2 can send it only the objects that do not already exist in its catalog.

Example:

```
(DATA
.
.
(SENDS
USSNANR.AUSTIN2
USSNANR.AUSTIN3
USSNANR.NEWYORK1
USSNANR.CHICAGO4
)
.
.
)
```

8.3.3 Problem Determination

If a particular object has a status of *SCHEDULED* or *IN PROGRESS* that does not reflect its true status, the existing version of the software object might be in a bad locked state. The result is that NetView DM/2 cannot build new versions of the object. NetView DM/2 always attempts to purge all previous locked requests when it builds new versions of software to be distributed. However, there are abnormal NetView DM/2 cases where locked objects require further manual intervention to correct locked NetView DM/2 files.

If during NetView DM/2 processing, messages indicate that NetView DM/2 failed because it could not remove a previous version of an object, try the steps listed below. If these steps fail to correct the problem, contact your IBM NetView DM/2 representative.

1. Check the NetView DM/2 message.dat file on the server and, if possible, on the client, to see whether any NetView DM/2 errors have been detected. If a NetView DM/2 error has occurred, take appropriate steps to report and correct the problem. Rebooting your system sometimes will temporarily correct the NetView DM/2 condition so that you can get your work done. Even if you take this route, go on to the next steps after rebooting.
2. Look at the request queue contents and install history to find the object name generated from NetView DM/2 processing. The name of the NetView DM/2 object is also in the messages. The locked entries can be identified by inspecting the NetView DM/2 request queue and seeing an entry for the object, where it is obvious that the entry is not being processed. At other times, a locked entry can be found by looking at the install history for clients that have install status other than INRU.
3. Temporarily undefine from the server the CC clients that involve the lock. Locked requests for undefined clients can sometimes be corrected by making the CC client unknown to the CC server. This step causes the NetView DM/2 CC client to stop running. Restart the client either manually or through another remote access to the machine.
4. If any requests are in the request queue, try to both purge and delete the request. (If you are using the GUI, steps 5 through 8 can be combined into one or two steps.)
5. Perform a Deleteit against the object's group name to remove install target information that might have been set previously. Do this for all client workstations with the /ws option.
6. Remove all install history for the object.
7. Redefine to the server the CC clients that were locked.
8. Try your initial NetView DM/2 request again.

8.3.4 NVBridge Utilities

TeamConnection provides a collection of utilities that can be combined into a user-defined build script to help automate customized forms of packaging steps. You can use these utilities instead of the *teamcpak* command to customize your distribution steps.

Note: These interfaces are program-sensitive interfaces used by NVBridge/2. As a result, they are likely to change and evolve from release to release. An IBM commitment that these interfaces will remain unchanged and compatible in future versions is not implied.

The following utilities are shipped with TeamConnection:

- ☐ FHPSTAT
- ☐ FHPOBDEL
- ☐ FHPOBMON
- ☐ FHPOBDIF
- ☐ FHPISCAT
- ☐ FHPICAT
- ☐ FHPUCAT
- ☐ FHPMCAT
- ☐ FHPVERIF
- ☐ FHPRQPUR
- ☐ FHPRQMON
- ☐ FHPTRVER
- ☐ FHPTRPUR

The only form of parameter checking that these utilities perform is verifying that the required parameters are specified and that the parameter list meets syntax requirements. These utilities assume that valid parameter values are passed to them.

To display the syntax of these utilities, type the name followed by a question mark. For example, type *FHPOBDEL ?* to see the syntax of this tool.

For more information about the individual utilities please refer to the *IBM TeamConnection for OS/2 Version 1 User's Guide*.

In this chapter we explain integrated problem tracking and change control and show how you can use them to manage and control your development process. We also cover the **configurable processes** that TeamConnection enables you to apply for problem tracking and change control and the various roles different people in the development organization assume. TeamConnection enables you to go from no control, through loose control, to very strict control by using the configurable processes.

One of the important functions of TeamConnection that distinguishes it from ordinary version control systems is its process management. As we mention in "Integrated Problem Tracking and Change Control" in topic 2.6, the processes involved in problem tracking and change control enable you to manage and control your development process. Whether you are a small or large development team, early in the development life cycle, ready for system test, or doing maintenance, TeamConnection lets you choose the level of control that you want to apply to your development process.

TeamConnection enables you to track reported problems and design changes and retain information about the life cycle of each. TeamConnection uses:

- A **defect** to record a reported problem
- A **feature** to record a proposed design change

The information recorded about defects and features enables you to report on who, what, when, why, and where modifications occur, where a particular defect or feature is in the development cycle, and where the release is in the development cycle. To control the behavior of *defects* and *features*, TeamConnection uses **component processes** (see "Planning the Component Process" in topic 3.7.1).

TeamConnection also enables you to control and record the changes made to your development data. It ties the process of managing defects and features to the change control process through the use of the track process. The track process is what we call a **release process** (see "Planning Your Release Processes" in topic 3.7.2). A release process determines to what extent part changes are tracked and the steps for integrating changed parts into a build. Release processes control the day-to-day work involved in fixing defects and implementing features, as well as building the product. If you use the **track** process, changes to managed objects must be tied to defects and features. Because changes are implemented in work areas, with the track process you can build, test, and ultimately commit individual changes to the release.

Component processes and release processes are configurable processes and, when combined, give you *integrated problem tracking and change control*. Figure 106 illustrates *problem tracking and integrated problem tracking and change control*.



Figure 106. Problem Tracking and Integrated Problem Tracking and Change Control

Subtopics

- 9.1 The Roles People Play
- 9.2 Working with Defects and Features
- 9.3 Change Control
- 9.4 Summary

9.1 The Roles People Play

TeamConnection has the strength to reflect the needs of an organization and the different roles within the organization.

TeamConnection maps users with certain roles by **authority groups** and **interest groups** (see "Access Lists" in topic 3.5.1 and "Notification Lists" in topic 3.5.2). The family administrator can modify these groups or create new ones to reflect the needs of the organization. Each group consists of actions usually performed by a particular type of user.

The following predefined authority groups come with TeamConnection:

- ☐ Projectlead
- ☐ Componentlead
- ☐ Releaselead
- ☐ Builder
- ☐ Developer+
- ☐ Writer+
- ☐ Developer
- ☐ Writer
- ☐ General
- ☐ Manager

Each group has predefined authorities to perform certain sets of TeamConnection actions. The family administrator can modify the mapping of groups or roles to TeamConnection actions or define new groups or roles.

When working with *integrated problem tracking and change control* it is very important to understand what the different authority groups are allowed to do and what their roles are. Before we explain that, however, let us review the commands that can be used for integrated problem tracking and change control.

Subtopics

9.1.1 Commands They Can Use

9.1.2 Actions They Can Perform

9.1.1 Commands They Can Use

The following commands are used for integrated problem tracking and change control:

Approval

The **approval** command is used to record approvers' opinions on approval records about proposed changes to parts in a release. This command can be used only for a work area in the approve state. The **approval** command provides greater control over changes made to releases as final deadlines approach.

Approval records are created automatically every time a work area is created for a release that has an approver list. Additional approval records can also be created for a work area, with the **approval** command, without changing the approver list associated with the release. The **approval** command can also be used to delete work area approval records or assign them to other users.

Owners of an approval record must indicate on it whether they accept or reject the changes proposed by the work area. An abstain option is available.

The state of the approval record controls whether the associated work area can move to the fix state. If the release process includes the approval process, when all approval records are in the accept or abstain state, the work area moves automatically to the fix state. If an approval record is in the reject state, the work area cannot move to the fix state.

Approver

The **approver** command is used to create entries on, and delete entries from, a release approver list. Each entry associates a user ID with a release, making the owner of the user ID an approver for any proposed changes to address defects or features in the specified release. The release approver list provides greater control over changes made to releases as final deadlines approach.

Every time a work area is created for a release to address a defect or a feature, approval records are created for each of the user IDs on the approver list associated with that release (provided that the release's process includes the approval subprocess). Each approval record refers to one defect or feature in one release and is owned by one approver. Approvers must use the **approval** command to accept or reject the proposed changes. Modifying an approver list does not change existing approval records.

Approval records that are accepted allow the work area to move to the fix state. If one or more approvers reject an approval record, the work area cannot move to the fix state.

Coreq

The **coreq** command is used to create and delete corequisite relationships between two or more work areas that are in the fix or integrate state. The work areas you identify as corequisites must all apply to the same release to be built together. Work areas defined as prerequisites by the TeamConnection product must also be built and committed together.

Identify corequisite relationships between work areas to indicate that work being done in one or more work areas is dependent on changes to parts associated with changes in another work area. These work areas must therefore be built together (committed together) so that the resulting code works correctly. This action ensures that a driver that includes one or more groups of *corequisite work areas* cannot be committed unless all of the work areas in the corequisite group are included in the driver.

Defect

The **defect** command is used to report problems by opening defects. It is also used to modify properties of defects, change the state of defects, and view information about defects.

When you open a defect, you become the originator of the reported defect. You must describe the problem you think has to be resolved and the primary component affected by the problem. By default the component owner is the owner of the defect unless it is not assigned to another user. The owner of the defect must respond to it by accepting it, returning it, or assigning it to a different component or user ID. If the design, size,

and review (*dsrDefect*) subprocess is included in the managing component's process, the defect owner must respond to it by designing it, returning it, or assigning it to a different component or user ID.

As the originator of the defect, you can cancel or reopen it if it is returned by the defect owner, and you can modify selected properties of a defect.

Driver

The **driver** command is used to:

- ☐ Create and delete drivers
- ☐ Commit the part changes related to drivers
- ☐ Extract the part tree represented by drivers
- ☐ Obtain information about existing drivers

A driver is a set of part changes for a release. To create a driver, you assign a name to it and relate it to a release. You then define a set of work areas as driver members. These work areas contain the parts that you want in a driver. If you create a driver, you become the driver owner by default; however, you can reassign ownership of the driver to another user.

To make permanent all part changes associated with the driver, you move the driver to the commit state. Before you can do this:

- ☐ All driver members must be in the integrate or commit state.
- ☐ All prerequisite and corequisite work areas must be included in the driver.
- ☐ You must have explicit access authority or be a superuser.

If you have explicit access authority, you can indicate when a driver is ready for formal testing by specifying that the driver is complete. This action changes the state of the associated work areas to the test state if an environment list exists for the release associated with the work areas. Otherwise, the work areas move to the complete state.

DriverMember

The **driverMember** command is used to specify the work areas you want to include in a given driver. Driver members are used only if the release has the driver process turned on. The work areas must be in the integrate state. A single work area can be a member of more than one driver. After a work area is committed in a driver, the other drivers in which it is a member ignore the committed work area.

By making a work area part of a driver, you associate the parts changed in relation to that work area with the specified driver. These parts must be members of the release associated with the driver. The work areas must also be members of the release associated with the driver.

You cannot create or delete driver members from a driver after the driver is committed.

Feature

The **feature** command is used to open requests for design changes or ideas for future functions. Also use this command to delete, modify properties of, change the state of, and view information about features.

The states a feature moves through depend on the TeamConnection subprocesses included in its associated component process. A component process can include the feature design, size, and review (*dsrFeature*) or *verifyFeature* subprocesses, or none at all.

When you open a feature, you become the originator of the feature. You must describe the proposed design change and name the primary component affected by the feature. The owner of the component you assign the feature to becomes the feature owner. If the *dsrFeature* subprocess is included in the component's process, the feature owner responds to the feature by moving it to the design state, returning it, or assigning it to a different component or user ID. If the *dsrFeature* subprocess is not included in the component's process, the feature owner either accepts the feature, returns it, or assigns it to a different component or user ID.

As the originator of the feature, you can cancel or reopen it if

it is returned by the feature owner. You can also modify selected properties of a feature. Originators of duplicate features are notified when the corresponding active feature is closed or canceled. Thus, they can either cancel or reopen the duplicate feature, as appropriate.

Fix

The **fix** command is used to create, delete, and reassign fix records and change the state of fix records.

Fix records are associated with work areas. A fix record reflects the status of all part changes made to resolve a defect or implement a feature for a work area and release in reference to one component. A work area has one or more fix records associated with it, depending on the number of components in which parts are changed. The component manages the parts to be changed in relation to the work area.

Each fix record is uniquely identified by a work area, a release, and a component. The owner of a fix record is, by default, the owner of the related component; however, this ownership can be reassigned by using the **-assign** action flag.

Each fix record refers to the part changes required within one component. The state of the fix record indicates the state of part changes for that component.

Fix records are created according to the sizing records of a feature or defect at the time a work area is created for the feature or defect. Fix records are created in the **notReady** state if the associated work area is in the **approve** state; otherwise, they are created in the **ready** state. Additional fix records are created if parts are changed and checked in to the TeamConnection product for a work area associated with a defect or a feature in a component for which there is no existing fix record. In this case, the fix record is in the active state. Active state means that part changes have been checked in to a work area associated with a defect or feature in the component. You can create fix records, using the **-create** action flag, if a work area is in the approve state or the fix state.

Use the **-complete** action flag to indicate that the part changes for the work area associated with a defect or feature within that component are completed. This moves the fix record to the complete state.

When all fix records for the work area are completed, the work area moves from the fix state to the integrate state. Use the **-activate** action flag to reactivate a fix record that is in the complete state if additional part changes are needed. You must change the work area state from integrate to fix before activating the fix records. You can make changes in the work area only when it is in the fix state.

If you decide that no part changes are required for a component that has a fix record, you can use the **-delete** action flag to delete the fix record from the associated work area.

Size

The **size** command is used to create, delete, and reassign sizing records for a defect or feature that is in the size state, or to indicate sizing information. A sizing record must be created explicitly by the defect or feature owner. A sizing record indicates the time and resources needed to resolve a defect or implement a feature in one component for a release. Each sizing record is uniquely identified by a defect or feature identifier, a component, and a release.

If you are the owner of the component in which the defect must be resolved or the feature must be implemented, you are also the owner of the sizing record by default. Sizing information must be entered as text on a sizing record.

All sizing records must be marked either with **accept** or **reject** in order to move the defect or feature from the size state to the review state. Work areas and fix records are created for all sizing records marked accept, when the defect or feature is accepted, when the track process is enabled for the release, and the design, size, and review process is enabled for the component. Otherwise, you must create the work area manually.

Test

The **test** command is used to indicate the results of an environment test on a test record associated with a work area.

If a release has an environment list, test records are created according to the entries in that list whenever a new work area is created for that release (provided that the release's process includes the test subprocess). Each test record includes the environment name and user ID specified on the release environment list, and the defect or feature identifier of the work area. The owner of the user ID is the tester who owns the test record.

Test records are activated (that is, they are moved to the ready state) when the associated work area moves to the test state and the proposed change is ready for environment testing. When results are entered for all of the environment test records associated with a work area, the state of that work area changes to complete. Even if you reject a test record, the work area changes to the complete state. Create another work area to address any changes still required.

Verify

The **verify** command is used to verify the resolution of defects or the implementation of features or to reassign ownership of existing *verification records*.

A verification record is created for the originator of a defect or a feature when the defect or feature is accepted and the component that manages the defect or feature has a process that includes the **defectVerify** or **featureVerify** subprocesses. Additional verification records are created for the originators of duplicate defects or features and attached to the active defect or feature. Defects can be specified as duplicates of features, and features can be specified as duplicates of defects.

Verification records become active when a defect or feature changes from the working state to the verify state. When results have been recorded for all of the verification records for a defect, and when all of that defect's work areas are complete, the defect changes from the verify state to the closed state. The same is true for a feature.

The defect or feature moves to the closed state even if you indicate unsuccessful results by marking your verification with **-reject**. In this case, you should open a new defect or feature to address the changes still required.

Workarea

The **workarea** command is used to create, modify, reassign, delete, freeze, refresh, and view information about a work area, and to change the state of a work area. A work area monitors the progress of changes to resolve a defect or implement a feature.

The states a work area moves through depend on the TeamConnection subprocesses included in the associated release process. A release process can include the track, approval, fix, driver, or test subprocesses, or none at all.

If the track process is turned on for the release, a work area must be associated with a defect or feature. The default name for these work areas is the name of the defect or feature.

You can also create a work area with a specified name. If the release does not have the track process, you must create the work area with a specified name. The user who creates the work area becomes the owner of the work area unless a different owner is specified when the work area is created.

You can also freeze or refresh your work area. You freeze a work area to save the current version of a part or parts in the work area by using the **-freeze** action. The version that you freeze represents a snapshot in time of the parts in your work area. You refresh your work area to verify that parts in the work area are the most current by using the **-refresh** action. When you refresh a work area, you get the most current view of all the parts in your work area from that source.

Changes associated with the work area are not visible to the release until you commit the work area. If your release does not have the driver process enabled, TeamConnection commits the work area implicitly through the **workarea -integrate** command.

If a defect or feature is linked to more than one release, multiple work areas exist for that defect or feature. The work

areas required for a defect or feature are created according to the accepted sizing records, when the defect or feature changes to the working state. Defect or feature owners can create additional work areas if the defect or feature is in the working state.

To determine the prerequisite and corequisite work areas for a particular work area, use the **workarea -check** command. By default, the **workarea -check** command lists all prerequisite and corequisite areas relative to the current state of the release. Specify the driver name to determine the prerequisite and corequisite work areas relative to an earlier committed driver. You will get a list of all prerequisite and corequisite work areas including any that were integrated after the specified driver was committed.

9.1.2 Actions They Can Perform

The following is an explanation of the actions TeamConnection's predefined authority groups can perform with regard to integrated problem tracking and change control:

Projectlead

A user with the authority of *Projectlead* is allowed to perform all actions on the following commands:

- ☐ **Approval**
- ☐ **Approver**
- ☐ **Coreq**
- ☐ **Defect**
- ☐ **Driver**
- ☐ **DriverMember**
- ☐ **Feature**
- ☐ **Fix**
- ☐ **Part**
- ☐ **Release**, except for **-prune**
- ☐ **Size**
- ☐ **Test**
- ☐ **Verify**
- ☐ **WorkArea**

A user with the authority of *Projectlead* can be a project leader for a large project that comprises different applications (releases). The project leader has the overall responsibility for the successful completion of the project but will probably rely on other users with authorities such as *Componentlead*, *Releaselead*, and *Builder* to lead the individual subprojects.

Componentlead

A user with the authority of *Componentlead* is usually a component owner and as such is responsible for the creation of *access lists* and *notification lists*. Typical users with the authority of *Componentlead* could be a design leader, database administrator, or document administrator.

A user with the authority of *Componentlead* is allowed to perform all actions on the following commands:

- ☐ **Coreq**
- ☐ **Defect**
- ☐ **Feature**
- ☐ **Part**, except for **-destroy**
- ☐ **Size**
- ☐ **Verify**

The *Componentlead* user is also allowed to:

- ☐ Create, delete, extract, or modify builders
- ☐ Create, delete, or modify parsers
- ☐ Accept, reconcile, or reject collision records
- ☐ Extract, freeze, or refresh drivers
- ☐ Assign or create fix records
- ☐ Extract releases
- ☐ Create, freeze, modify, or refresh work areas

Releaselead

A user with the authority of *Releaselead* is probably a team leader for a subproject and responsible for the creation of a new release when development starts on a new baseline, parsers and builders are created for the release, a new release is linked to other releases if needed, and work done in work areas is integrated with the release.

A user with the authority of *Releaselead* is allowed to perform all actions on the following commands:

- ☐ **Approval**
- ☐ **Approver**
- ☐ **Coreq**
- ☐ **Fix**
- ☐ **Part**, except for **-destroy**
- ☐ **Release**, except for **-delete** and **-recreate**
- ☐ **Test**
- ☐ **WorkArea**, except for **-commit** and **-test**

The *Releaselead* user is also allowed to:

- ☐ Create, delete, extract, or modify builders
- ☐ Create, delete, or modify parsers
- ☐ Accept, reconcile, or reject collision records

- ☐ Extract, freeze, or refresh drivers

Builder

The main responsibility of a user with *Builder* authority is to create the builders necessary to do the final builds, *commit* and *complete* drivers, and do the driver builds. A driver represents the whole system or application, but before it is integrated into the release.

A user with the authority of *Builder* is allowed to perform all actions on the following commands:

- ☐ **Coreq**
- ☐ **Driver**
- ☐ **DriverMember**
- ☐ **Fix**, except **-assign** and **-delete**
- ☐ **Part**, except for **-destroy**

The *Builder* user is also allowed to:

- ☐ Create, delete, extract, or modify builders
- ☐ Create, delete, or modify parsers
- ☐ Accept, reconcile, or reject collision records
- ☐ Extract, freeze, or refresh drivers
- ☐ Extract releases
- ☐ Create sizing records
- ☐ Check, complete, create, fix, freeze, or refresh work areas

Developer+

A user with the authority of *Developer+* is a member of the development team and might be responsible for some of the functions in the application, such as: the user interface, file access, or any other logic of the application. A user with the authority of *Developer+* is allowed to do almost all actions related to the **Part** command except for: **part -destroy** and in some instances **part -unlock** (you can not *unlock* a part that has been locked by another user).

The *Developer+* user is also allowed to:

- ☐ Create builders and parsers
- ☐ Design, size, and review defects and features
- ☐ Create or delete corequisites
- ☐ Extract, freeze, and refresh drivers
- ☐ Create, delete, complete, or reactivate fix records
- ☐ Extract releases
- ☐ Create sizing records
- ☐ Freeze or refresh work areas

Writer+

A user with the authority of *Writer+* is probably a technical writer or editor. This user can do almost the same actions related to the **Part** command as the *Developer+* except for **part -connect** and **part -disconnect**.

The *Writer+* user is also allowed to:

- ☐ Extract builders
- ☐ Design, size, and review defects and features
- ☐ Create or delete corequisites
- ☐ Extract drivers
- ☐ Create, delete, complete, or reactivate fix records
- ☐ Extract releases
- ☐ Create sizing records

Developer

A user with the authority of *Developer* could probably be a contractor or a member of the development team with very limited authority. The *Developer* user is allowed to do the following:

- ☐ Extract builders
- ☐ Add, build, check in, check out, extract, rename, and touch parts
- ☐ Extract releases

Writer

A user with *Writer* authority has the same authority as a *Developer* user. This user is probably an editor or contractor working with documentation.

You might notice that we have not mentioned the *General* and *Manager* authority groups. The *General* authority group is not involved in any real work at all and, apart from the base authority, can only view various

things within the family. The same is true for the *Manager* authority group. But there is also a *Manager* notification group, which allows somebody belonging to the *Manager* group to be notified about:

- ☐ Abstain, accept, or reject of approval records
- ☐ Accept, assign, close, modify, open, reopen, return of defects and features
- ☐ Abstain, accept, or reject of test records
- ☐ Abstain, accept, or reject of verification records

9.2 Working with Defects and Features

Any TeamConnection user can open a defect or feature. Each defect and feature must be opened against a specific component within the component hierarchy. The defect or feature can be reassigned to a more appropriate component within the hierarchy if necessary. The owner of the component to which it is assigned automatically becomes the owner of the defect or feature. This ownership can also be reassigned.

Figure 107 shows the different relationships for *defects* and *features*. The figure tells us the following:

- ☐ One *user* owns many defects and/or features.
- ☐ One *component* manages many defects and/or features.
- ☐ Many *design records* belong to one defect or feature.
- ☐ Many *sizing records* belong to one defect or feature.
- ☐ One defect or feature can define many *work areas*.
- ☐ One *verification record* belongs to one defect or feature.



Figure 107. Defect and Feature Relationships

Subtopics

9.2.1 Defect and Feature States

9.2.2 A Sample Feature

9.2.1 Defect and Feature States

Depending on the subprocesses that have been configured, defects and features move through different states during their life cycles (see Figure 108). Valid states are *open*, *design*, *size*, *review*, *working*, *verify*, *closed*, *returned*, and *canceled*.

The owner is responsible for analyzing a defect or feature once it is opened. He or she can then return it if it is not valid or feasible, reassign it to another user, or accept it for resolution. The owner can return a defect or feature for any reason. Only the originator can cancel a defect or feature. If a defect or feature is returned, the originator may want to record more information about the problem or enhancement and then reopen it. If the defect or feature does not relate to the component for which it was reported, the owner can reassign it to a more appropriate component within the hierarchy.

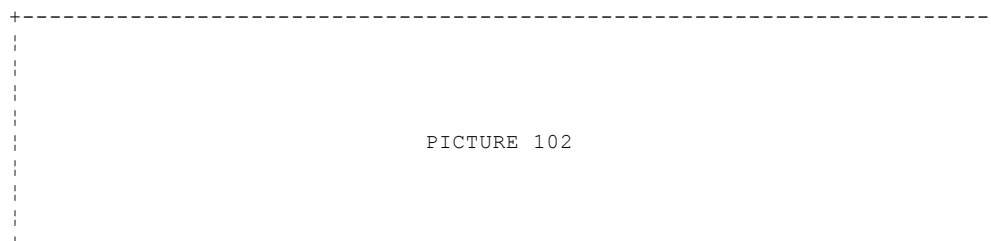


Figure 108. Defect and Feature States

The possible states for defects and features are:

Open state

When you open a defect or feature, it is in the open state and you are considered the originator. You assign the defect or feature to a component. The owner of this component becomes the feature or defect owner and is responsible for managing its resolution. The component in which you open a defect or feature should be one that manages the parts affected by the enhancement or problem. Use the component descriptions and the structure of your family's hierarchy to find the most appropriate component. If you open a defect or feature in an inappropriate component, the component owner can reassign it. The defect or feature owner is responsible for implementation, and the originator is responsible for verifying that the defect or feature is resolved correctly.

Returned state

A defect or feature owner can return a defect or feature to its originator. You can return a feature or defect from the open, design, size, or review state if you decide that the defect or feature is not feasible or not valid. You can return a defect or feature in the working state only if it has no associated work areas. If there are associated work areas, you must cancel them before you can return the defect or feature. When you return a defect or feature, add your reason for returning it so that the originator and any other users can evaluate why you believe it is not feasible or not valid.

Canceled state

A feature or defect in the open or returned state can be canceled only by its originator or by a superuser. A canceled defect or feature remains inactive unless it is reopened by the originator.

Design state

If the component to which a defect or feature is assigned includes the design, size, and review subprocess, defects or features in the open or returned state move to the design state. In this state, the proposed change is designed, and a description of the design change is entered. The owner must describe the design change before the defect or feature can move to the next state.

Size state

Defects or features move to this state after the owner enters design information. In this state, users can create a sizing record for each release that contains parts affected by the enhancement or problem. A sizing record identifies the work that is required for and the resources affected by the defect or feature. The owner of the component that is referenced in the sizing record is the owner of the sizing record. The owner is responsible for entering information about the amount of work that is required to implement the feature or resolve the problem. The sizing record owner can reject the sizing record

if it does not affect the specified component. After all sizing records are either accepted or rejected, the defect or feature moves to the review state or returns to the design if more design information is needed.

Review state

Defects or features move to this state after they have been sized. In this state, the design text and sizing records are reviewed to determine the feasibility of the proposal. The owner can do one of the following:

- ☐ Accept the defect or feature if all design and sizing records are acceptable. This moves the defect or feature to the working state.
- ☐ Return the defect or feature to the originator if all design and sizing records are not acceptable. If necessary, the originator can reopen a defect or feature.
- ☐ Move the defect or feature back to the design state if design modifications are needed.

Working state

Defects or features move to this state when the owner accepts the defect or feature when it is in the:

- ☐ **Review** state if the DSR subprocess is included in the process
- ☐ **Open** state if the DSR subprocess is not included in the process

When you accept a defect or feature, you accept the responsibility of resolving it. A defect or feature might require changes in more than one release. If the component includes the DSR subprocess, these releases were identified during the size state, and TeamConnection created a work area for each identified release. If the component does not include the DSR subprocess, you will have to create a work area.

When the first work area moves to the complete state, the defect or feature automatically moves to the verify state or closed state. However, if a release is specified when the defect is created, the defect moves to the verify state when all work areas (there can be multiple work areas for the defect) for the release are integrated.

Verify state

Defects and features go through the verify state only if their component includes the verify subprocess. When a defect or feature is accepted, TeamConnection creates a verification record. This record lets the originator:

- ☐ Accept the fix if the resolution was satisfactory
- ☐ Reject the fix if not satisfied with the resolution
- ☐ Abstain if unable to assess the resolution

The defect or feature cannot be closed until the originator takes one of the above actions. After the defect or feature is in the verify state, it cannot return to the working state. Therefore, if you reject a defect or feature because you are not satisfied with the resolution, open a new defect or feature to propose any necessary changes.

Closed state

The closed state is the final state of a defect or feature. If verification records exist for a defect or feature, the defect or feature goes to the closed state after all verification records are in the accept, reject, or abstain state and all work areas are in the complete state. If verification records do not exist, the defect or feature can go directly from the working state to the closed state. You cannot re-open a defect or feature that is in the closed state. If the defect or feature was not resolved correctly, you must open a new defect or feature to address the necessary changes.

Figure 109 shows an *action-state diagram* for defect and feature work areas. Here we can see which actions we need to take to move a defect or feature from one state to another. In this diagram we also show you the actions for parts and work areas.

Note: Figure 109 only shows actions for defects; however, the actions for features are the same, so you can use the diagram for both defects and features.

+-----+

PICTURE 103

Figure 109. Action-State Diagram

Subtopics

9.2.1.1 Design, Size, Review

9.2.1.1 Design, Size, Review

Once a defect or feature has been accepted, the actual implementation must be designed and sized, after which a review will be held.

The design phase must be performed by users familiar with the product or area affected by the defect or feature. Design text is recorded within the defect or feature and can be supplemented by other users until a complete design exists.

During the sizing phase, sizing records are created. Sizing records identify the resources required to resolve the defect or implement the feature. There is a sizing record for each component and release pair affected by the defect or feature.

Each owner of a component referenced in a sizing record has to evaluate the impact of the defect or feature on the files managed by that component (impact analysis). If the objects managed by a component are affected by the defect or feature, this fact is recorded by accepting the sizing record and adding sizing information. If the objects are not affected, this fact is recorded by rejecting the sizing record.

Once the resolution has been designed and the resources identified, the proposal has to be reviewed. At this time the need for additional design work may be identified.

Features and defects that are not feasible can be returned to the originator. Any returned defect or feature can be reopened, if necessary, by the originator.

9.2.2 A Sample Feature

In this section we present an example of opening a feature for a release that has no tracking process defined, but with a component process of *preship* defined for one of the components. Having the *preship* component process defined means that a defect or feature has to go through all defect and feature states. Thus we have to go through the design, size, review, and verify states before we can close the feature.

Subtopics

- 9.2.2.1 Opening a Feature
- 9.2.2.2 The Design State
- 9.2.2.3 The Size State
- 9.2.2.4 The Review State
- 9.2.2.5 Accepting the Feature
- 9.2.2.6 The Working State
- 9.2.2.7 The Verify State
- 9.2.2.8 The Closed State

9.2.2.1 Opening a Feature

We will open a feature related to this book. While we were writing this chapter, we realized that we had to create some more figures using Freelance Graphics. We had already created the first three figures in this chapter and we thought that an *action-state diagram* (see Figure 109 in topic 9.2.1) would fit in nicely in the section "Defect and Feature States" in topic 9.2.1.

Use the **Open Feature** window (see Figure 110) to open a new feature. The feature originator is the user who opens a feature. The feature owner is, by default, the owner of the component to which the feature is assigned. Any user within the family can open a feature against any component in the hierarchy.

The following fields and push buttons are available in the **Open Feature** window:

Component

Type the name of the component to which you want to assign the feature. If you specified a component on the Environment page of the Settings notebook, TeamConnection displays that component's name in this field. You can change the name if you want to add a feature to a different component. The owner of the component becomes the feature owner. The feature owner is responsible for managing the resolution of the feature.

Remarks

Type a description of the problem directly into this field.

Edit

You can also select the **Edit** push button to type lengthy remarks or to insert text from a file.

Abstract

You can type a brief description of the feature. If you do not type information in this field, TeamConnection uses the first 63 characters from the Remarks field for the abstract.

Name

You can type up to 15 alphanumeric characters that identify the feature. If you do not specify a name, the system assigns one.

Reference

You can specify another defect or feature, or even something outside TeamConnection, that is related to the feature you are opening.

Prefix

Select the prefix that describes the type of feature that you found.

OK

Processes the information that you typed and closes the window

Apply

Processes the information you typed and leaves the window open

Cancel

Does not process the information that you typed and closes the window

Import

Imports text into the field that has cursor focus

Help

Displays help information about the window

Note: Your family administrator can configure additional fields for the window. TeamConnection help is not available for any additional fields that your family administrator creates.

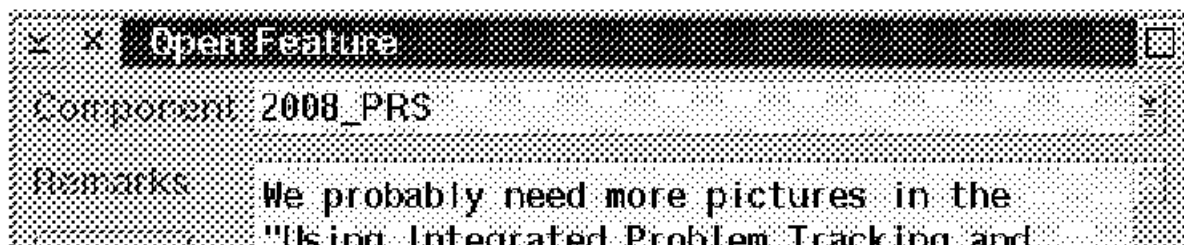


Figure 110. Open Feature Window

Command Line Interface Syntax

The following is the command line interface syntax for opening a feature:

```
teamc feature -open -remarks Text -component Name -family Name  
[-name Name] [-prefix Name]* [-reference Name]  
[-abstract Text] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-open	Opens a feature. (A unique identifier is generated by TeamConnection to identify the new feature unless you specify an identifier using the optional -name flag.)
-remarks <i>Text</i>	Describes the change being requested, the actual design for the feature, or the reason for modifying or changing the state of the feature. After you issue a command that adds remarks, you cannot change the remarks. To move a feature to the size state, you must have entered some design text using the -remarks flag within the -design action.

Attribute Flag and Argument	Description
-component <i>Name</i>	Specifies the name of the component when opening or assigning a feature. The environment variable is not used for feature -assign (environment variable: TC_COMPONENT).
-family <i>Name</i>	Specifies the family for which this feature is being opened (environment variable: TC_FAMILY).

-name <i>Name</i>	Specifies the feature identifier. Up to 15 alphanumeric characters are allowed for user-generated feature IDs. The TeamConnection product checks the uniqueness of the ID. If the ID already exists in the TeamConnection product, the action fails and you receive a message indicating that the identifier is not unique. You must then enter a new identifier or allow the TeamConnection product to generate one.
-prefix <i>Name</i>	Categorizes features by type. This value precedes the feature identifier in report output.
-reference <i>Name</i>	Assigns a value, name, or keyword to a feature
-abstract <i>Text</i>	Lets users enter concise text to summarize a feature. Up to 63 characters are allowed. This text appears in reports and notification messages. If this flag is not specified when you open a feature, the first 63 characters or the text up to the first new-line character of the -remarks flag serves as the abstract.

Attribute Flag and Argument	Description
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this


```
|-----| command. |-----|
+-----+
```

In our case, the command syntax would look like this:

```
teamc Feature -open -component 2008_PRS
-remarks 'We probably need more pictures in the
"Using Integrated Problem Tracking and Change Control" chapter.'
-name 2008FU-1 -prefix f
```

After having succeeded in opening the feature, we get an information window as shown in Figure 111.

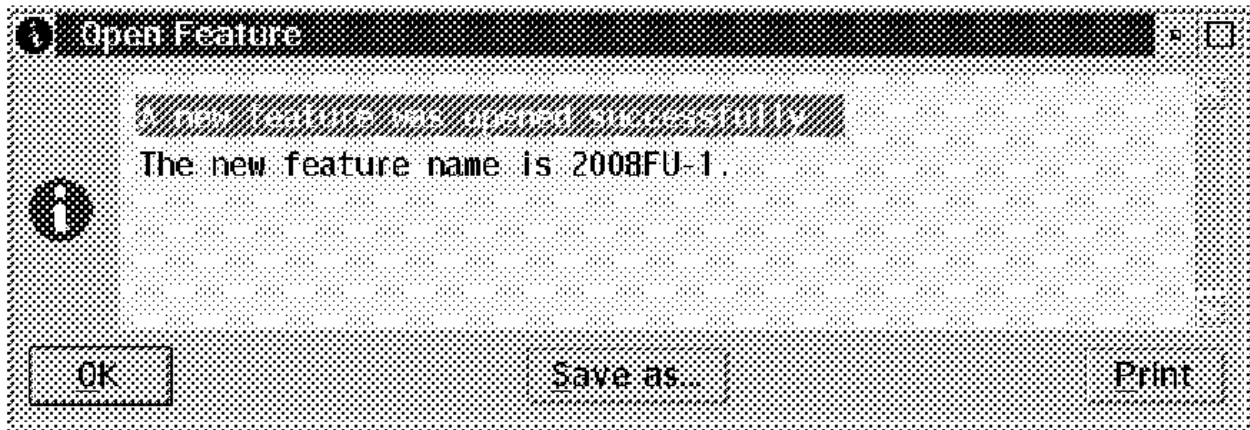


Figure 111. Open Feature Information Window

Once the feature has been opened, you can work with it in the **TeamConnection - Features** window (see Figure 112). By selecting the feature you want to work with and depending on the *state* of the feature, you can choose the following items from the pop-up menu:

View

Displays the **View Feature Information** window that enables you to view information about a feature

Process configurations

Displays the **View Feature Process Configurations** window that enables you to see information about the process that is configured for the component to which the specified features are assigned

Design

Displays the **Design Features** window that enables you to move a feature or enter design remarks in a feature

Size

Displays the **Size Features** window that enables you to move a feature from the design state to the size state so that the effort to implement the feature can be sized

Review

Displays the **Review Features** window that enables you to move a feature from the size state to the review state so that approvers can review the proposed feature implementation and its sizing information

Accept

Displays the **Accept Features** window that enables you to move a feature from the open or review state to the working state

Verify

Displays the **Verify Features** window that enables you to move a feature from the working state to the verify state

Return

Displays the **Return Features** window that enables you to move a

feature to the returned state

Reopen

Displays the **Reopen Features** window that enables you to move a feature that is in the returned or canceled state to the open state

Add remarks

Displays the **Add Remarks to Features** window that enables you to add comments about a feature. You can add comments at any time.

Cancel

Displays the **Cancel Features** window that enables you to move features to the canceled state

Create sizing records

Displays the **Create Sizing Records** window that enables you to create a sizing record.

Create work areas

Displays the **Create Work Areas** window that enables you to create a new work area to follow a defect or feature in a particular release.

Verification records >

Select the following to accept, reject, or abstain from verifying verification records:

Accept

Displays the **Accept Verification Records** window that enables you to complete a verification record by approving it

Reject

Displays the **Reject Verification Records** window that enables you to complete a verification record by rejecting it

Abstain

Displays the **Abstain From Verification** window that enables you to complete a verification record without expressing either approval or disapproval

Modify >

Select one of the following to modify the owner, originator, component, name, or properties of features:

Owner

Modifies the owner of features

Originator

Modifies the originator of features

Component

Modifies the component of features

Name

Modifies the name of features

Properties

Modifies the properties of features

Show >

Select *Show* to display the change history, sizing records, verification records, approval records, fix records, test records, driver records, or work areas.

In many cases, the information that TeamConnection displays when you select one of the options listed below depends on items you have selected in the corresponding panel. If you have not selected any objects in the window, TeamConnection cannot determine what you want to display; it therefore brings up the **Filter** window so that you can specify what you want displayed.

Change history

Displays the change history information for releases, drivers, and parts

Sizing records

Displays the sizing records for the selected features

Verification records

Displays the verification records for the selected features

Approval records

Displays the approval records for the selected features

Fix records

Displays the fix records for the selected features

Test records

Displays the test records for the selected features

Driver members

Displays the driver members for the selected features

Work Areas

Displays the **Work Area** window that enables you to work with work areas.

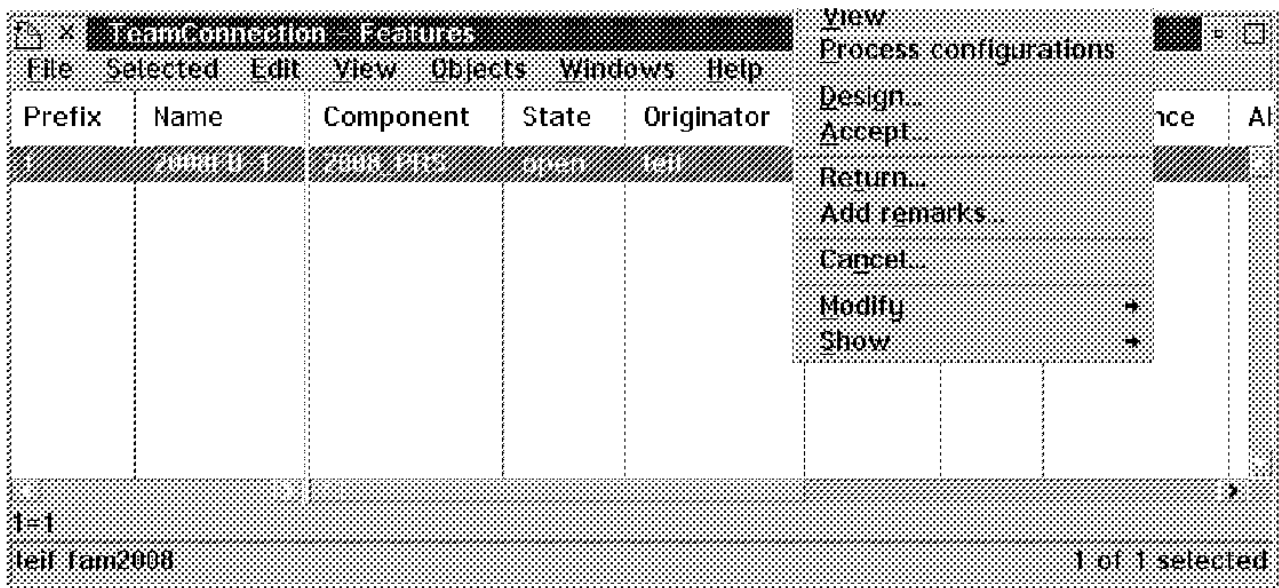


Figure 112. The Features Window

As you can see in Figure 112, the choices available from the pop-up menu in our case are:

- ☐ View
- ☐ Process configurations
- ☐ Design
- ☐ Accept
- ☐ Return
- ☐ Add remarks
- ☐ Cancel
- ☐ Modify
- ☐ Show

As we have the DSR subprocess defined for the component to which the feature was assigned, we will have to go through the design, size, and review states. Thus we will continue with "The Design State" in topic 9.2.2.2. If we had not had the DSR subprocess defined, we would have continued with "Accepting the Feature" in topic 9.2.2.5.

9.2.2.2 The Design State

When the feature (or defect) is in the **design** state, we will propose the design of our implementation. Thus we will verbally describe our design by entering text in the *Remarks* field.

Use the **Design Features** window (see Figure 113) to:

- ☐ Move a feature to the **design** state.
- ☐ Enter design remarks for a feature in the **design** state.

The feature must be in one of the following states:

- ☐ Open
- ☐ Design
- ☐ Size
- ☐ Review
- ☐ Returned

If you move a feature to the **design** state from the **review** state, you must add a remark to the sizing records.

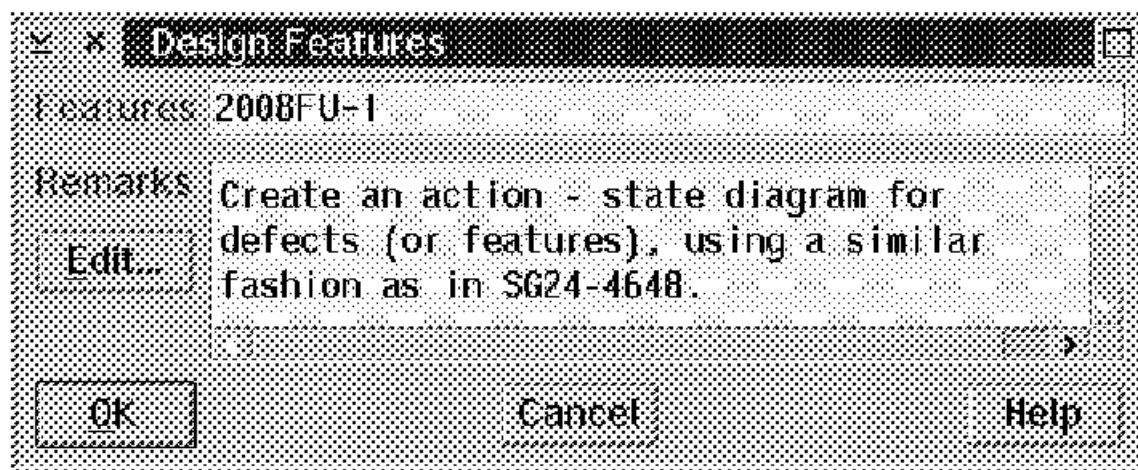


Figure 113. Design Features Window

Command Line Interface Syntax

The following is the command line interface syntax for moving a feature to the **design** state:

```
teamc feature -design Name ... -family Name [-remarks Text]
                [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-design Name ...	Moves features to the design state or specifies design text. Features can move to the design state from the open, returned, design, size, or review state.
-family Name	Specifies the family for which this feature is being opened (environment variable: TC_FAMILY)
-remarks	Describes the change

Text	being requested, the actual design for the feature, or the reason for modifying or changing the state of the feature. After you issue a command that adds remarks, you cannot change the remarks. To move a feature to the size state, you must have entered some design text using the -remarks flag within the -design action.
-become Name	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command.

In our case, the command syntax would look like this:

```
teamc Feature -design 2008FU-1
-remarks 'Create an action-state diagram for
defects (or features), using a similar
fashion as in SG24-4648.'
```

If we look in the **TeamConnection - Features** window (see Figure 114), we see that in the *State* column the state has changed from **open** to **design** (compare with Figure 112 in topic 9.2.2.1). By using the *action-state diagram* in Figure 109 in topic 9.2.1, we see that there are three actions that we can take. We can either:

- ☐ Provide additional **design** text for the feature
 - ☐ **Size** the feature
- or
- ☐ **Return** the feature

TeamConnection - Features					Process configurations	
File	Selected	Edit	View	Objects	Windows	Help
Prefix	Name	Component	State	Originator		
1	2008FU-1	2008FU-1	design	CU	Design...	nce
					Size	A
					Return...	
					Add remarks...	
					Modify	→
					Show	→

Figure 114. Features Window

In our case, we decide to move on to the *size* state, because we are satisfied with our design.

9.2.2.3 The Size State

Use the **Size Features** window (see Figure 115) to move the feature (or defect) to the **size** state.

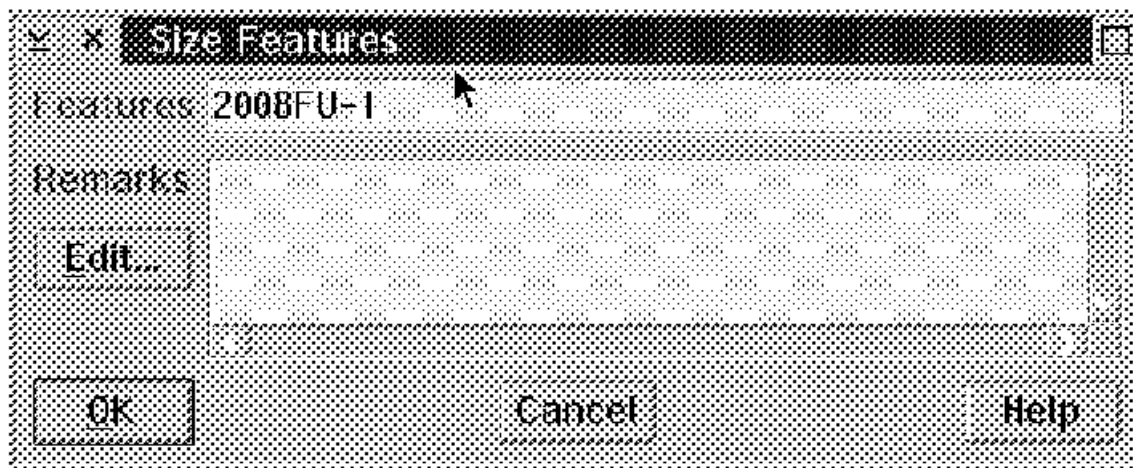


Figure 115. Size Features Window

Command Line Interface Syntax

The following is the command line interface syntax for moving a feature to the **size** state:

```
teamc feature -size Name ... -family Name [-remarks Text]
                [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-size Name ...	Moves features from the design state to the size state for sizing. Design text must first be entered using feature -design -remarks .
Note: The family, remarks, become, and verbose attribute flags and their arguments are explained in "The Design State" in topic 9.2.2.2.	

In our case, the command syntax would look like this:

```
teamc Feature -size 2008FU-1
```

When a feature is in the **size** state, you can create *sizing records*. You can create one sizing record per release and component pair. The feature owner has authority to create sizing records. By default, the owner of the sizing record is the owner of the component for which it was created. After all sizing records have been accepted or rejected, you can move the feature to the **review** state.

Use the **Create Sizing Records** window (see Figure 116) to create a sizing record for the defects or features you own. The defect or feature owner must create the sizing record. The sizing record indicates the time and resources needed to resolve a defect or implement a feature in one component for a release. A defect or feature name, a component, and a release uniquely identify each sizing record. To create a sizing record, ensure that the process associated with the component includes the DSR subprocess.

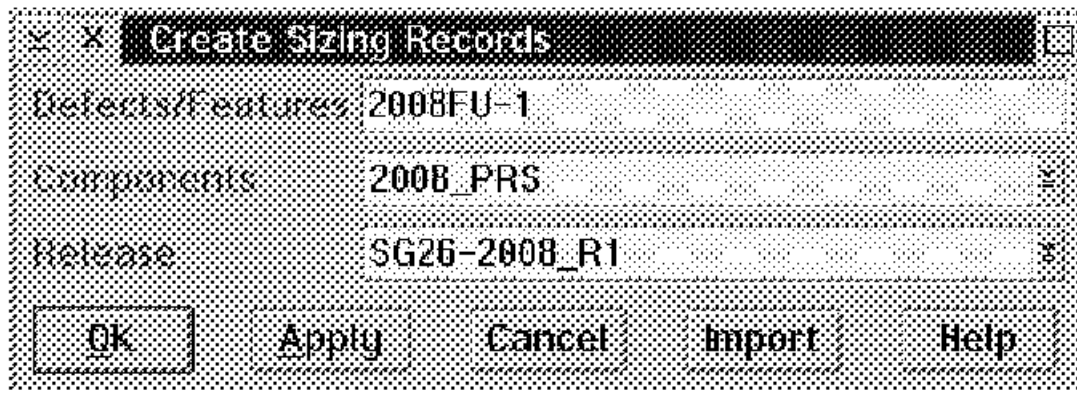


Figure 116. Create Sizing Records Window

Use the **TeamConnection - Sizing Records** window (see Figure 117) to view sizing information for the defect or feature you specify. You get to the **TeamConnection - Sizing Records** window by selecting the specific feature in the **TeamConnection - Features** window and from the pop-up menu select **Show > Sizing records**. Sizing records indicate the time and resources needed to resolve one defect or to implement one feature in one component for one release.

The owner of the defect or feature must create the sizing records. You can create sizing records only when a defect or feature is in the **size** state. By default, the owner of a sizing record is the owner of the component to which the sizing record applies. After the sizing record owner accepts or rejects each sizing record, the owner can move the defect or feature from the **size** state to the **review** state.

TeamConnection automatically creates fix records for a defect or feature for each sizing record in the **accept** state when the defect or feature is accepted.



Figure 117. TeamConnection - Sizing Records Window

Use the **Accept Sizing Records** window (see Figure 118) to size the effort involved in fixing the defect or implementing the feature. If you access the **Accept Sizing Records** window by selecting a defect or feature and using the selected pull-down or by using the pop-up menu (by selecting the defect or feature using the right mouse button), TeamConnection puts the correct data in all fields but sizing.

Enter the sizing information and select **OK** to accept the sizing record.

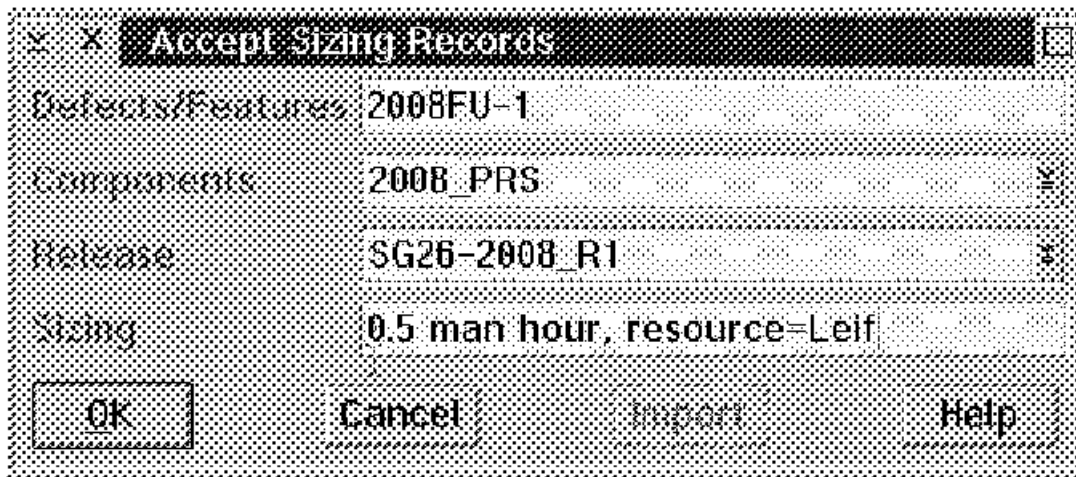


Figure 118. Accept Sizing Records Window

When all sizing records for a defect or feature have been either accepted or rejected, you can move the defect or feature to the **review** state. Fix records are created for all sizing records in the **accept** state after the defect or feature is accepted.

After having accepted the *sizing record*, we can now move the feature to the **review** state.

9.2.2.4 The Review State

Use the **Review Features** window (see Figure 119) to move a feature from the **size** state to the **review** state so that a formal review of the proposed feature resolution and its sizing information can occur. All sizing records for the feature must be accepted or rejected.

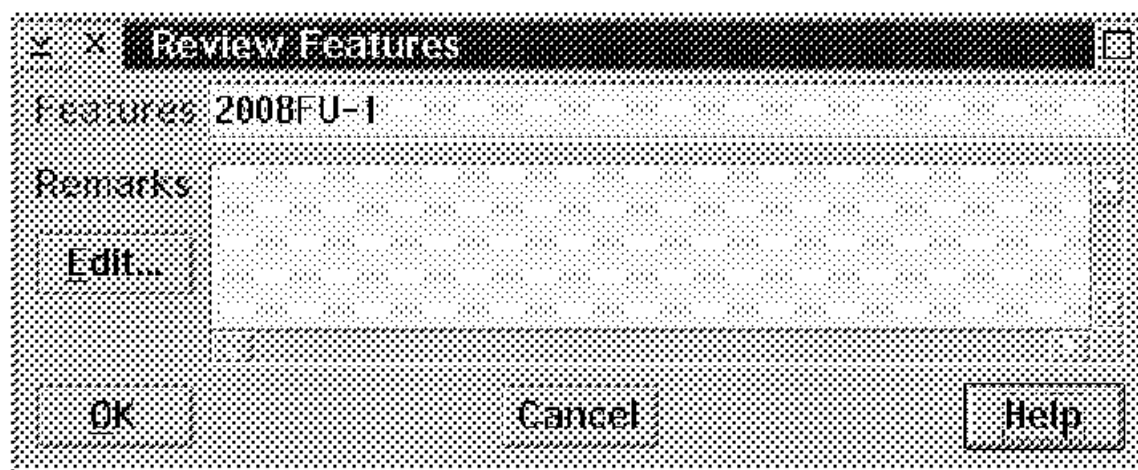


Figure 119. Review Features Window

Command Line Interface Syntax

The following is the command line interface syntax for moving a feature to the **review** state:

```
teamc feature -review Name ... -family Name [-remarks Text]
               [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-review Name ...	Moves features from the size state to the review state so that the proposed design implementation and sizing information can be reviewed.
Note: The family, remarks, become, and verbose attribute flags and their arguments are explained in "The Design State" in topic 9.2.2.2.	

In our case, the command syntax would look like this:

```
teamc Feature -review 2008FU-1
```

Defects or features move to the **review** state after they have been sized. In this state, the design text and sizing records are reviewed to determine the feasibility of the proposal. The owner can do one of the following:

- Accept the defect or feature if all design and sizing records are acceptable. This moves the defect or feature to the **working** state.
- Return the defect or feature to the originator if all design and sizing records are not acceptable. If necessary, the originator can reopen a defect or feature.
- Move the defect or feature back to the **design** state if design modifications are needed.

In our case, we believe that our design is accurate and we can therefore move the feature to the **working** state. In the next section we show you how to move a feature (or defect) to the **working** state.

9.2.2.5 Accepting the Feature

To move the feature to the **working** state, you have to accept the feature. If the process for the component includes the DSR subprocess, the feature must be in the **review** state. If the process for the component does not include the DSR subprocess, the feature must be in the **open** or **returned** state.

TeamConnection creates a verification record in the **notReady** state. If the release includes the track subprocess, TeamConnection creates a work area for each sizing record for the feature after the feature is accepted.

Use the **Accept Features** window (see Figure 120) to move a feature (or defect) from the **open** or **review** state to the **working** state so that formal work can begin on it.

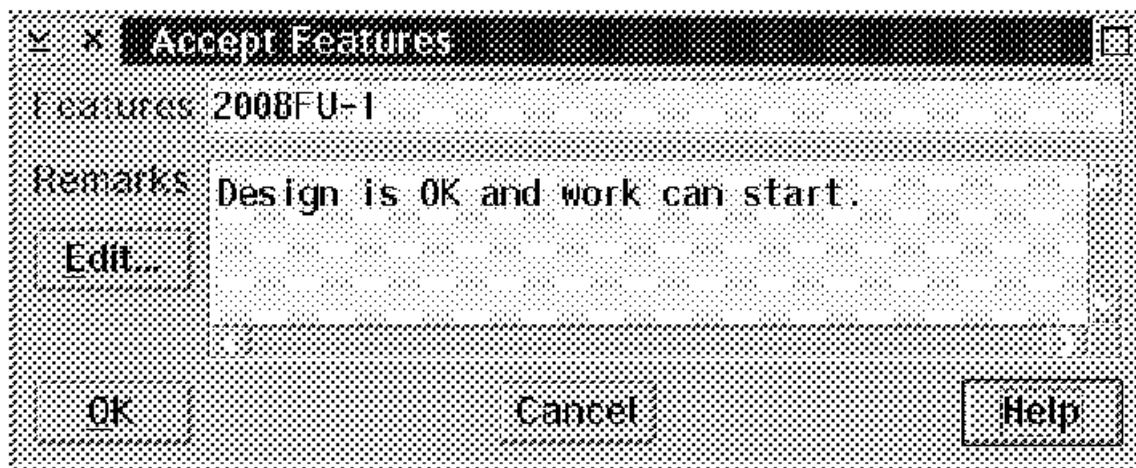


Figure 120. Accept Features Window

Command Line Interface Syntax

The following is the command line interface syntax for accepting a feature:

```
teamc feature -accept Name ... -family Name [-remarks Text]
          [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-accept Name ...	Accepts features, in the open or review state, depending on the subprocess configuration of the component so that problems can be resolved.
Note: The family, remarks, become, and verbose attribute flags and their arguments are explained in "The Design State" in topic 9.2.2.2.	

In our case, the command syntax would look like this:

```
teamc Feature -accept 2008FU-1
-remarks 'Design is OK and work can start.'
```

When you accept a defect or feature, you accept the responsibility of resolving it. A defect or feature might require changes in more than one release. If the component includes the DSR subprocess, these releases were identified during the **size** state, and TeamConnection created a work area (only if any of the release processes are defined) for each identified release. If the component does not include the DSR subprocess, you will have to create a work area yourself.

As we now have accepted the feature, we can now start the formal work on implementing the feature.

9.2.2.6 The Working State

Defects or features move to the **working** state when the owner accepts the defect or feature when it is in the:

- ☐ **Review** state if the DSR subprocess is included in the process
- ☐ **Open** state if the DSR subprocess is not included in the process

If any of the release subprocesses is defined for the release, a work area will be created for the defect or feature, and any changes to parts in the release can be done only through this defect or feature work area. Depending on which subprocess has been defined, specific actions have to be taken to move the work area from state to state (as explained in "Change Control" in topic 9.3).

As no release subprocess has been defined for our release, the defect or feature will not affect the changes that we are making to the parts in the release, and we only use the defects or features to track problems or implementations. We could actually have used TeamConnection to track any problems or enhancements, not necessarily problems or enhancements related to parts kept in the family. For example, we could have used TeamConnection to track the progress of problems or enhancements in a project regarding building a house.

9.2.2.7 The Verify State

Use the **Verify Features** window (see Figure 121) to manually move the feature (or defect) to the **verify** state when a work area does not exist and the work on the feature (or defect) is completed. The feature must be in the **working** state.

If work areas exist for a feature, TeamConnection automatically moves the feature from the **working** state to the **verify** state when the first work area (in any release) associated with that feature is in the **complete** state. Any verification records associated with the feature are moved from the **notReady** state to the **ready** state.

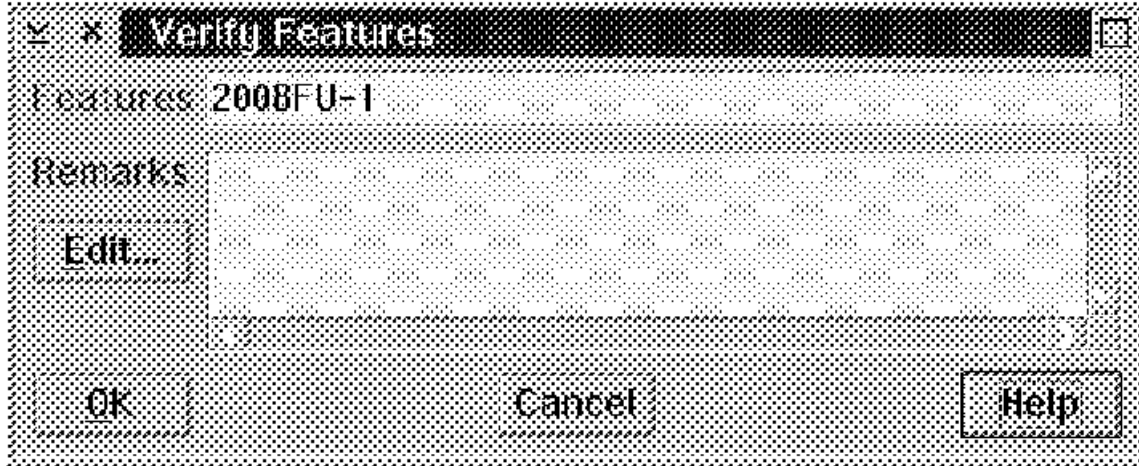


Figure 121. Verify Features Window

Command Line Interface Syntax

The following is the command line interface syntax for moving a feature to the **verify** state:

```
teamc feature -verify Name ... -family Name [-remarks Text]
               [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-verify Name ...	Moves features from the working state to the verify state
Note: The family, remarks, become, and verbose attribute flags and their arguments are explained in "The Design State" in topic 9.2.2.2.	

In our case, the command syntax would look like this:

```
teamc Feature -verify 2008FU-1
```

Defects and features go through the **verify** state only if their component includes the verify subprocess. When a defect or feature is accepted, TeamConnection creates a verification record. This record lets the

originator:

- ☐ *Accept* the fix if the resolution was satisfactory
- ☐ *Reject* the fix if not satisfied with the resolution
- ☐ *Abstain* if unable to assess the resolution

The defect or feature cannot be closed until the originator takes one of the above actions. After the defect or feature is in the **verify** state, it cannot return to the **working** state. Therefore, if you reject a defect or feature because you are not satisfied with the resolution, open a new defect or feature to propose any necessary changes.

Use the **Accept Verification Records** window (see Figure 122) to complete a verification record by approving it. TeamConnection creates a verification record for the originator of a defect or feature when the defect or feature is accepted, if the component to which the defect or feature is assigned includes the verify subprocess.

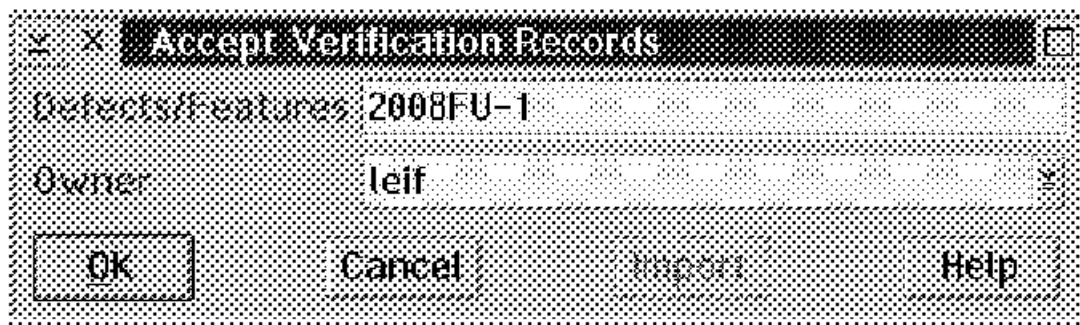


Figure 122. Accept Verification Records Window

Command Line Interface Syntax

The following is the command line interface syntax for accepting a verification record:

```
teamc verify -accept { -defect Name ... -feature Name ... }
                  -family Name [-tester Name] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-accept	Indicates successful verification of the defect resolution or feature implementation
-defect Name ...	Specifies the defect identifiers associated with the verification records
-feature Name ...	Specifies the feature identifiers associated with the verification records
-family Name	Specifies the family for which this feature is being opened (environment variable: TC_FAMILY)
-tester Name	Identifies the owner of the verification record if you are performing the verification for

someone else

Attribute flag and argument	Description
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command.

In our case, the command syntax would look like this:

```
teamc Verify -accept -defect 2008FU-1 -tester leif
```

Use the **Reject Verification Records** window to reject a verification record. This action completes the verification record. If you reject a verification record, the fix record is closed after all verification records have been *accepted*, *rejected*, or *abstained*.

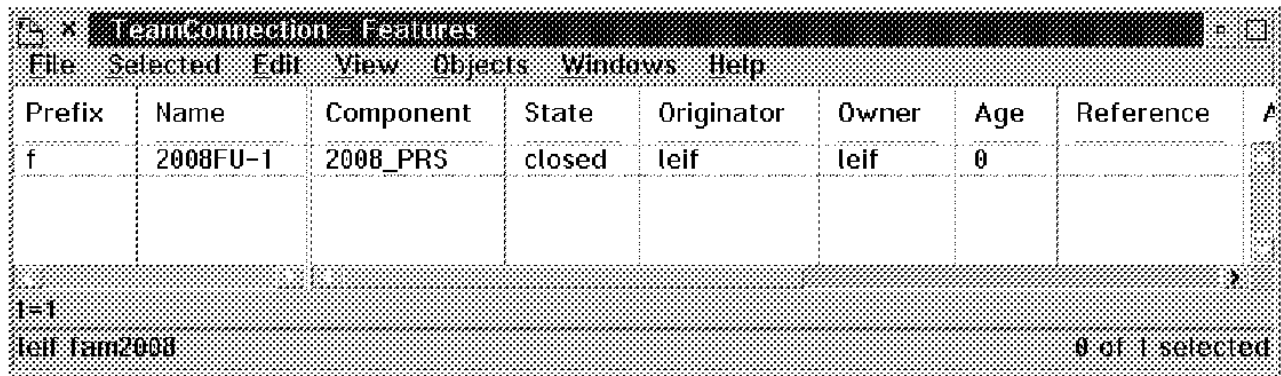
Use the **Abstain From Verification** window to complete a verification record without expressing either approval or disapproval.

As we are satisfied with the new picture, we *accept* the verification record for the feature and this takes us and the feature to the **closed** state.

9.2.2.8 The Closed State

When all verification records for a defect or feature have been either accepted, rejected, or abstained, and all of the work areas for that defect or feature are complete, the defect or feature automatically changes from the **verify** state to the **closed** state. If you *reject* a verification record because you are not satisfied with the fix, you must open a new defect to address the problem. Rejecting the verification record does not prevent the defect or feature from closing.

Figure 123 shows the **TeamConnection - Features** window after the verification record has been accepted. The state has changed to **closed**, which indicates that this particular feature is now closed. If, for some reason, you are not satisfied with the resolution, you can always open a new defect or feature to propose any necessary changes.



The screenshot shows a window titled "TeamConnection - Features". It has a menu bar with "File", "Selected", "Edit", "View", "Objects", "Windows", and "Help". Below the menu bar is a table with the following columns: "Prefix", "Name", "Component", "State", "Originator", "Owner", "Age", and "Reference". The table contains one row of data: "f", "2008FU-1", "2008_PRS", "closed", "leif", "leif", "0", and an empty reference field. At the bottom of the window, there is a status bar that reads "1=1" and "leif fam2008" on the left, and "0 of 1 selected" on the right.

Prefix	Name	Component	State	Originator	Owner	Age	Reference
f	2008FU-1	2008_PRS	closed	leif	leif	0	

1=1
leif fam2008 0 of 1 selected

Figure 123. TeamConnection - Features Window after Feature Has Been Closed

9.3 Change Control

TeamConnection provides different processes for components and releases. The processes you choose depend on how tightly you want to control changes and how you want to handle defects and features. Your choices will vary depending on where you are in your current development cycle. You can change your processes during a development effort to reflect different phases. For example, you might do the following:

- During the requirements gathering phase, you create a component that manages the requirements documentation. You want minimal defect or feature processing against the parts managed by this component, so you select a process, such as *prototype*, that is not strict. The release would also follow a relaxed process, such as *prototype*.
- After the requirements are settled and design work begins, you want to control changes to the requirements data but not to the rapidly evolving design data. For the requirements component, you change to a process that includes review and verification, such as the *default* process. You also create a new component to manage the design documentation and select a process that is not strict. You continue to use the *prototype* process for the release.
- When coding begins, you change the process for the design component to one that includes review and verification, such as *development*. You also create a new component to manage the code files, and you select a process that is not strict, such as *prototype*. You also change to a release process, such as *development*, that tracks the resolution of defects and features.
- After all the code files managed by a given component pass unit test, you change that component's process to one that includes review and verification, such as *default*. You also change the release process to one with tight control, such as *track_full*, so that you can carefully manage code changes.
- Ninety days before your delivery date, you change all of the components to a very stringent process, such as *preship*, to ensure that all new features or defects are reviewed for their impact on the delivery schedule.

Just as defects and features move through different states during their life cycle, so do defect and feature work areas and drivers. In the next two sections we explain the different states and how they affect your development work.

Subtopics

9.3.1 Work Area States

9.3.2 Driver States

9.3.3 Using the Preship Process

9.3.1 Work Area States

Depending on which release subprocesses have been configured, defect and feature work areas move through different states during their life cycles (see Figure 124). Valid states are **approve**, **fix**, **integrate**, **commit**, **test**, and **complete**.



PICTURE 118

Figure 124. Defect and Feature Work Area States

The possible states for defect and feature work areas are:

Approve state

When a work area is created, it goes to the **approve** state if the *approval subprocess* is included in the release process. TeamConnection creates an **approval record** for each user on the release's approver list. Approvers indicate their evaluation of the changes in their approval records:

- ☐ Accept that work should continue
- ☐ Abstain if unable to assess if work should continue
- ☐ Reject if work should not continue

When all approval records are marked as abstain or accept, the work area automatically goes to the **fix** state. If any approval record is marked as reject, the state remains at **approve**.

Fix state

If there are no approver lists for a release, work areas for the release begin in the **fix** state. While the work area is in this state, parts are checked out from the work area, changes are made to those parts, and builds are done to verify the accuracy of the changes.

A **fix record** monitors the part changes within a single component. Fix records provide a mechanism for reviewing all part changes that apply to components before integrating those changes with changes made for other defects and features. TeamConnection creates fix records for features or defects when existing sizing records are accepted. If a fix record does not already exist for the component, TeamConnection creates one when a part managed by that component is checked in to the server. If a defect or feature owner needs to create the work area, he or she can create fix records at the same time. Existing fix records go to the **active** state when a part is checked in with reference to the work area.

When all necessary part changes within the specified component have been made, the changes can be reviewed or inspected. The fix record owner is responsible for this review. When the fix record owner is satisfied that the part changes made within that component are complete and ready for integration with other parts in the release, he or she marks the fix record as complete. When all existing fix records for a work area are complete, the work area automatically moves to the **integrate** state. If the *fix subprocess* is not included, fix records are not created, and the work area must be moved to the **integrate** state explicitly.

If the release does not include the tracking process, a work area moves to the **complete** state after a work area integrate action.

Integrate state

If the *fix* and *driver subprocesses* are included in the release process, the work area automatically moves to the **integrate** state when all fix records are complete. If necessary, the work area owner can force a work area to the **integrate** state, provided that no part changes are associated with the work area. You can add work areas in the integrate state to an existing driver as driver members if the *driver subprocess* is configured. All work areas in the **integrate** state do not have to be added to the same driver.

If the *driver subprocess* is included in the release process,

drivers can be explicitly created at any time. Each driver automatically moves to the **integrate** state when the first work area is added to it as a driver member. If all work areas are removed from the driver, the driver automatically returns to the **working** state. Committing a driver commits all work areas included as driver members and all parts that were changed in reference to those work areas.

The work area stays in the **integrate** state until the driver in which it is a member is committed. The work area owner can force a work area to the **commit** state, provided that no part changes are associated with the work area. If the release process does not include the driver subprocess, the integrated work area moves to the **test** state if the test subprocess is included, or the complete state if the *test subprocess* is not included.

If the *driver subprocess* is not configured, you can explicitly commit the work area.

Commit state

The work area moves automatically to the **commit** state when the driver to which it belongs is committed. At this point, all parts that were changed in this release to resolve the feature or defect are committed. The work area remains in the **commit** state until the driver to which it belongs is completed.

Test state

A work area goes through the **test** state if the release includes the *test subprocess*. When the associated driver moves to the **complete** state or when a work area is committed without a driver, the work area moves to the **test** state. The driver is then ready for formal testing in the specified environments. Any existing **test records** for the work area move to the **ready** state when the work area moves to the **test** state. The work area stays in the **test** state until all test records move from the **ready** state.

Complete state

The **complete** state is the final state of a work area. If the *test subprocess* is not included in the release process, the work area moves directly to this state when the associated driver is completed or when the work area is explicitly integrated.

When a work area is completed, the feature or defect associated with that work area automatically moves to the **verify** or **complete** state. The defect does not leave the **working** state until the work area for that release is completed.

Figure 125 shows an *action-state diagram* for defect and feature work areas. Here you can see which actions you have to take to move a work area from one state to another. The diagram also shows the actions for parts.

Note: Figure 125 only shows actions for defect and feature work areas without the driver process being defined.

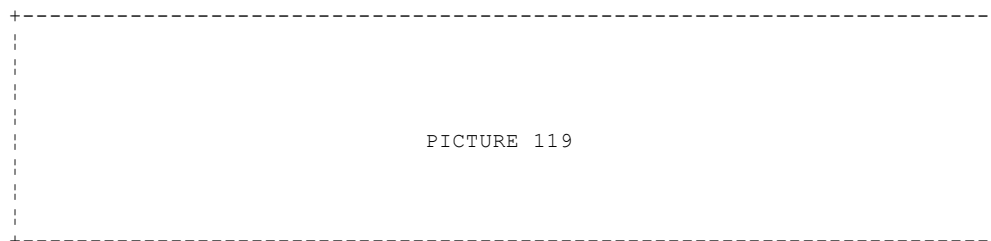


Figure 125. Action-State Diagram for Work Areas

9.3.2 Driver States

If both the *track* and the *driver subprocess* have been defined, multiple defects and features are integrated to what is called a *driver* before they are committed to a release. The defect and feature work areas are added as **driver members** to the driver (see Figure 126). This allows changes to be built and tested together before committing them to a release. In other words, a driver represents a mini subset of a release, but before it is seen in the release itself.



Figure 126. Adding Driver Members

A *driver* also moves through different states during its life cycle (see Figure 127). Valid states are **working**, **integrate**, **commit**, and **complete**.

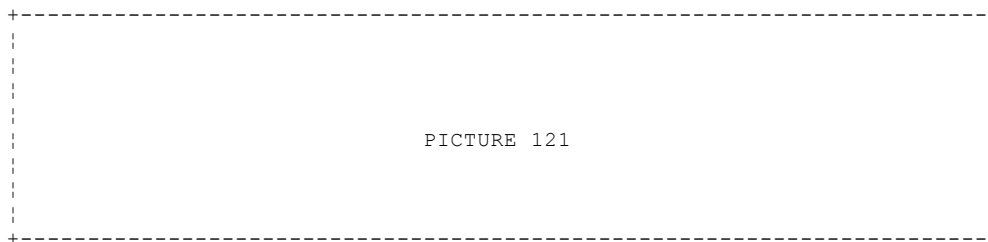


Figure 127. Work Area and Driver States

The possible states for drivers are:

Working state

The **working** state is the initial state of a driver. While the driver is in this state, it is not associated with any work areas and therefore contains no part changes.

Integrate state

Work areas can be added to drivers as driver members when the driver is in the **working** or **integrate** state and the work area is in the **integrate** state. As soon as the first driver member is added, the driver automatically moves to the **integrate** state. You can extract the driver when it is in the **integrate** state; however, only those parts that were changed in reference to driver members are extracted. This is referred to as extracting a *delta part tree*.

Commit state

TeamConnection commits only a successfully built driver. Committing a driver changes the state of the driver to **commit**. You can, however, explicitly commit the driver.

When a driver moves to the **commit** state, all work areas that are included as driver members also move to the **commit** state. When a work area is in the **commit** state, all part changes associated with the work area become the current version of the release and are visible to all users of the release.

A committed driver can be extracted as a *full part tree* as well as a *delta part tree*. A full part tree is the part structure of all parts within the release.

Complete state

The **complete** state is the final state of a driver. In this state, the driver is ready for formal testing in the specified environments. If the release includes the *test subprocess*, the work areas that are included as driver members move to the **test** state. Otherwise, the work areas move to the **complete** state.

If the component includes a *verify subprocess*, verification records are sent to the originators of any defects or features that were addressed in the completed driver.

9.3.3 Using the Preship Process

In the sections that follow we use a games application called *Serata* to illustrate change control. Our *Serata* application is now in the *preship* process (we are only a couple of weeks away from first delivery). If we look at Figure 24 in topic 3.7.2 in "Planning Your Release Processes" in topic 3.7.2, we see that the release *preship* and *track_full* processes are the two processes that contains all subprocesses. Thus if the release contains any of these two processes, the defect or feature work areas and drivers associated with this release have to go through all of the defined states. In addition the two processes apply the most stringent problem tracking and change control.

Figure 128 shows the state transitions for defects and features, work areas, and drivers.

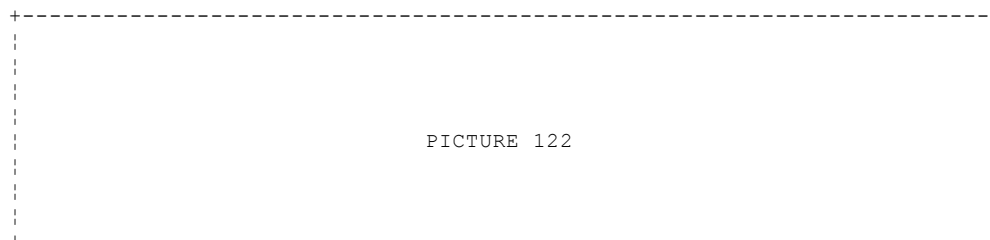


Figure 128. Defect and Feature, Work Area, and Driver State Transitions

Figure 128 can be very helpful in showing where defects and features, work areas, and drivers are supposed to be and where they can go from there. But the *action-state diagram* in Figure 129 also shows you which actions are related to which states. When you start using TeamConnection, it is very easy to get lost when you start applying more stringent process control, but the *action-state diagrams* that we provide in this book will be very helpful to you.

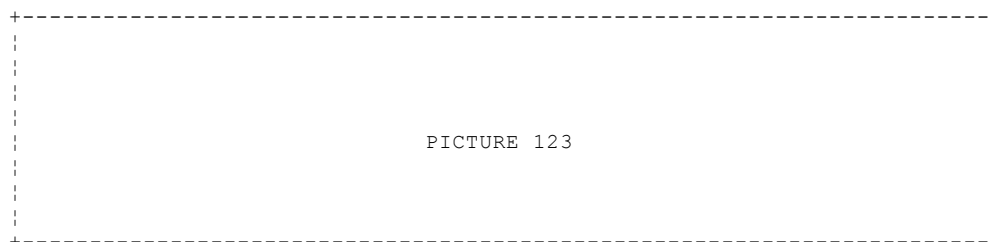


Figure 129. Action-State Diagram for the Driver Subprocess

As our component for the *Serata* application also has the *preship* process defined, we will apply the most stringent problem tracking and change control that TeamConnection allows. We start our example by opening a *defect*.

Subtopics

- 9.3.3.1 Opening a Defect
- 9.3.3.2 Moving a Defect to the Design State
- 9.3.3.3 Moving a Defect to the Size State
- 9.3.3.4 Moving a Defect to the Review State
- 9.3.3.5 Accepting the Defect
- 9.3.3.6 The Defect in the Working State
- 9.3.3.7 The Work Area in the Approval State
- 9.3.3.8 The Work Area in the Fix State
- 9.3.3.9 Integrating the Work Area
- 9.3.3.10 Creating a Driver and Adding a Driver Member
- 9.3.3.11 Committing the Driver
- 9.3.3.12 Completing the Driver
- 9.3.3.13 Completing Test Records
- 9.3.3.14 Completing the Verification Records

9.3.3.1 Opening a Defect

A defect is a TeamConnection object that monitors and controls changes made to fix a problem with a project or product. As you and your team members make changes, the state of the defect changes. The state of a defect indicates its progress along the path of a process or subprocess. The state of a defect can limit further changes by other users.

We have just discovered that one of our include files (*board.h*) in the Serata application contains an error. The error has to be corrected, so we have to open a defect associated with the *Serata-C++* component. We can do this in several ways:

- ☐ By selecting **Actions > Defects > Open...** in the **TeamConnection - Tasks** window
- ☐ By selecting either **Open defect...** from the pop-up choices or **Selected > Open defect...** in the **TeamConnection - Components** window
- ☐ By selecting **Selected > Open...** in the **TeamConnection - Defects** window

Use the **Open Defect** window (see Figure 130) to open a defect using the GUI. The **Open Defect** window contains the following fields:

Component

Type the name of the component to which you want to assign the defect. If you specified a component in the Environment page of the Settings notebook, TeamConnection displays that component's name in this field. You can change the name if you want to add a defect to a different component. The owner of the component becomes the defect owner. The defect owner is responsible for managing the resolution of the defect.

Remarks

Type comments directly into this field. You can also select the **Edit** push button to type lengthy remarks or insert text from a file.

Abstract

Type a brief description of the problem. If you do not type information in this field, TeamConnection uses the first 63 characters from the Remarks field for the abstract.

Name

Type up to 15 alphanumeric characters that identify the defect. If you do not specify a name, the system assigns one.

Release

Type the name of the release in which the defect is to be fixed. Usually, this is the release in which you found the defect. You can use this release information when querying for defects.

Driver

Type the driver in which the defect was found, if applicable.

Environment

Type the environment in which the defect was found. You can then use the environment information when querying for defects. See your family administrator for a list of the possible environments.

Reference

Type another defect or feature, or even something outside TeamConnection, that is related to the defect you are opening.

Prefix

Type the prefix that describes the type of defect that you found.

Severity

Type the severity code that best describes the seriousness of the defect.

Symptom

Type the symptom that best describes the characteristics of the defect you found.

Phase found

Type the development phase that best describes the phase in which you found the defect.

Note: Your family administrator can configure additional fields for this window, which you can then use, for example, for additional impact analysis. TeamConnection help is not available for any additional fields that your family administrator creates.

You might notice that the **Open Defect** window contains more fields than the **Open Feature** window (see Figure 110 in topic 9.2.2.1).

The screenshot shows the 'Open Defect' window with the following fields and values:

Component	Serata-C++
Remarks	Beta test failed. Timing problem.
Abstract	
Name	d960614
Release	Serata-R1
Driver	R1D1
Environment	OS2
Reference	
Prefix	d
Severity	2
Symptom	test_failed
Phase found	beta_test

Buttons at the bottom: OK, Apply, Cancel, Import, Help.

Figure 130. Open Defect Window

After having clicked on the **OK** button, we get an information window (see Figure 131) that informs us that the defect was successfully opened.



Figure 131. Open Defect Information Window

Command Line Interface Syntax

The following is the command line interface syntax for opening a defect:

```
teamc defect -open -remarks Text -component Name -family Name
               [-name Name] [-prefix Name]* [-environment Name]
               [-severity Name]* [-reference Name] [-symptom Name]*
               [-phaseFound Name]* [-driver Name] [-abstract Text]
               [-release Name] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-open	Opens a defect. A unique identifier is generated by TeamConnection to identify the new defect, unless you specify an identifier using the optional -name flag.
-remarks <i>Text</i>	Describes the change being requested, the actual design for the defect, or the reason for modifying or changing the state of the defect. After you issue a command that adds remarks, you cannot change the remarks (that is, you cannot use defect -modify to change the remarks). To move a defect to the size state, you must have entered some design text using the -remarks flag within the -design action.

Attribute Flag and Argument	Description
-component <i>Name</i>	Specifies the name of the component when opening or assigning a defect. The environment variable is not used for defect -assign environment variable: TC_COMPONENT).
-family <i>Name</i>	Specifies the family for which this command is being issued (environment variable: TC_FAMILY)
-name <i>Name</i>	Specifies the defect identifier. Up to 15 alphanumeric characters are allowed for user-generated defect IDs. TeamConnection checks the uniqueness of the ID. If the ID already exists in TeamConnection, the action fails, and you receive a message indicating that the identifier is not unique. You must then enter a new identifier or let TeamConnection generate one.
-prefix <i>Name</i>	Identifies a prefix that categorizes the defect by type. This value precedes the defect identifier in report output.
-environment <i>Name</i>	Specifies the environment where a defect was discovered, for example, the OS/2 environment
-severity <i>Name</i>	Specifies the severity of the problem for which the defect was opened
-reference <i>Name</i>	Assigns a value, name, or keyword to a defect
-symptom <i>Name</i>	Specifies the symptom associated with the defect
-phaseFound <i>Name</i>	When opening or modifying a defect, specifies the development phase in progress when the defect was discovered
-driver <i>Name</i>	Specifies the driver in which the defect

was discovered

Attribute Flag and Argument	Description
-abstract <i>Text</i>	Lets users enter concise text to summarize a feature. Up to 63 characters are allowed. This text appears in reports and notification messages. If this flag is not specified when you open a feature, the first 63 characters or the text up to the first new-line character of the -remarks flag serves as the abstract.
-release <i>Name</i>	Specifies a particular release to trigger the defect verification process when the work area for this release moves to the complete state (environment variable: TC_RELEASE)
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command.

In our case, the command syntax would look like this:

```
teamc Defect -open -component Serata-C++
-remarks 'Beta test failed. Timing problem.'
-name d960614 -release Serata-R1 -driver R1D1 -environment OS2 -prefi d
-severity 2 -symptom test_failed -phaseFound beta_test
```

Once the defect has been opened, you can work on it in the **TeamConnection - Defects** window (see Figure 132). By selecting the defect you want to work with and depending on the *state* of the defect, you can choose the following from the pop-up menu:

View

Displays the **View Defect Information** window that enables you to view information about a defect

Process configurations

Displays the **View Defect Process Configurations** window that enables you to review information about which process and subprocesses are configured for the component to which the specified defects are assigned

Design

Displays the **Design Defects** window that enables you to move a defect to the design state or to enter design remarks for a defect

Size

Displays the **Size Defects** window that enables you to move a defect from the design state to the size state, so that the effort to resolve the defect can be sized

Review

Displays the **Review Defects** window that enables you to move a defect from the size state to the review state so that a formal review of the proposed defect resolution and its sizing information can occur

Accept

Displays the **Accept Defects** window that enables you to move a defect from the open or review state to the working state

Verify

Displays the **Verify Defects** window that enables you to move a defect from the working state to the verify state

Return

Displays the **Return Defects** window that enables you to move a defect to the returned state

Reopen

Displays the **Reopen Defects** window that enables you to move a defect that is in the returned or canceled state to the open state

Add remarks

Displays the **Add Remarks to Defects** window that enables you to add comments about a defect. You can add comments at any time.

Cancel

Displays the **Cancel Defects** window that enables you to move defects to the canceled state

Create sizing records

Displays the **Create Sizing Records** window that enables you to create a sizing record

Create work areas

Displays the **Create Work Areas** window that enables you to create a new work area to follow a defect or feature in a particular release

Verification records >

Select one of the following to accept, reject, or abstain from verifying verification records:

Accept

Displays the **Accept Verification Records** window that enables you to complete a verification record by approving it

Reject

Displays the **Reject Verification Records** window that enables you to complete a verification record by rejecting it

Abstain

Displays the **Abstain From Verification** window that enables you to complete a verification record without expressing either approval or disapproval

Modify >

Select one of the following to modify the owner, originator, component, name, or properties of defects:

Owner
Modifies the owner of defects

Originator
Modifies the originator of defects

Component
Modifies the component of defects

Name
Modifies the name of a defect

Properties
Modifies the properties of defects

Show >

Select *Show* to display the sizing records, verification records, approval records, fix records, test records, driver records, or work areas.

In many cases, the information that TeamConnection displays when you select one of the options listed below depends on items you have selected in the corresponding panel. If you have not selected any objects in the window, TeamConnection cannot determine what you want to display; it therefore brings up the **Filter** window so that you can specify what you want displayed.

Change history
Displays the change history information for releases, drivers, and parts

Sizing records
Displays the sizing records for the selected defects

Verification records
Displays the verification records for the selected defects

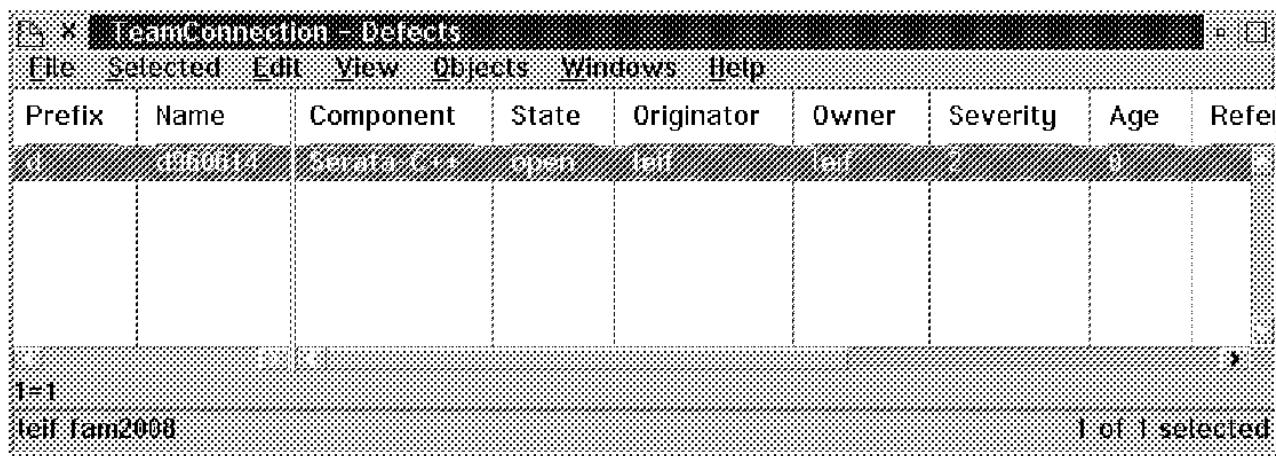
Approval records
Displays the approval records for the selected defects

Fix records
Displays the fix records for the selected defects

Test records
Displays the test records for the selected defects

Driver members
Displays the driver members for the selected defects

Work areas
Displays the **Work Area** window that enables you to work with work areas



The screenshot shows a window titled "TeamConnection - Defects" with a menu bar (File, Selected, Edit, View, Objects, Windows, Help) and a table of defects. The table has columns: Prefix, Name, Component, State, Originator, Owner, Severity, Age, and Reference. One defect is listed with the following values: Prefix: 0, Name: def00111, Component: Service P, State: open, Originator: fail, Owner: fail, Severity: 2, Age: 0, and Reference: . Below the table, it shows "1 of 1" and "1 of 1 selected".

Prefix	Name	Component	State	Originator	Owner	Severity	Age	Reference
0	def00111	Service P	open	fail	fail	2	0	.

1 of 1
fail fam2008 1 of 1 selected

Figure 132. TeamConnection - Defects Window

The defect is now in the **open** state (as you can see in Figure 132). Because the defect is in the **open** state, the following choices are available from the pop-up menu:

- ☐ View
- ☐ Process configurations
- ☐ Design
- ☐ Accept
- ☐ Return
- ☐ Add remarks
- ☐ Cancel
- ☐ Modify
- ☐ Show

As we have the DSR subprocess defined for the component to which the defect was assigned, we will have to go through the **design**, **size**, and **review** states. The procedures are the same as those described for a feature in "The Design State" in topic 9.2.2.2, "The Size State" in topic 9.2.2.3, and "The Review State" in topic 9.2.2.4. Therefore we do not describe in detail the different windows and commands needed to take a defect through these states (see "Working with Defects and Features" in topic 9.2). If we had not had the DSR subprocess defined, we would have moved the defect to the **working** state by accepting the defect as described in "Accepting the Feature" in topic 9.2.2.5.

9.3.3.2 Moving a Defect to the Design State

Use the **Design Defects** window (see Figure 133) to:

- ☐ Move a defect from the **open** state to the **design** state
- ☐ Enter design remarks for a defect that is in the **design** state

The defect must be in one of the following states:

- ☐ Open
- ☐ Design
- ☐ Size
- ☐ Review
- ☐ Returned

If you move a defect to the **design** state from the review state, you must add a remark to the sizing records.

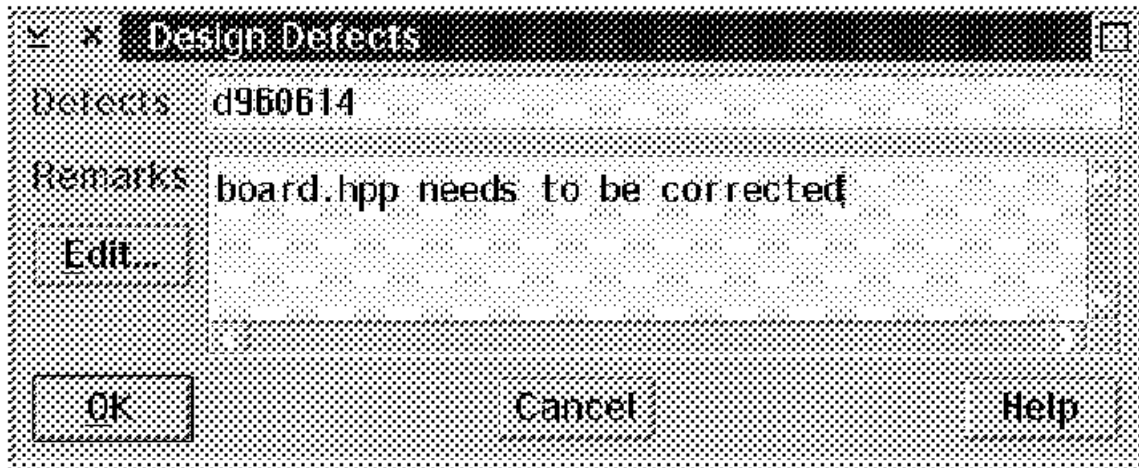


Figure 133. Design Defects Window

Command Line Interface Syntax

The following is the command line interface syntax for moving a defect to the **design** state:

```
teamc defect -design Name ... -family Name [-remarks Text]
                [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-design	Moves defects to the design state or specifies design text. Defects can move to the design state from the open , returned , size , or review state.
-family <i>Name</i>	Specifies the family for which this feature is being opened (environment variable: TC_FAMILY)
-remarks <i>Text</i>	Describes the change being requested, the actual design for the defect, or the reason for modifying or changing the

TeamConnection at Large
Moving a Defect to the Design State

state of the defect.
After you issue a
command that adds
remarks, you cannot
change the remarks
(that is, you cannot
use **defect -modify**
to change the
remarks). To move a
defect to the **size**
state, you must have
entered some design
text using the
-remarks flag within
the **-design** action.

-become *Name*

Identifies the user
ID from which you
want to issue
TeamConnection
commands, if the
user ID differs from
your login. You
assume the access
authority of the
user ID you specify
(environment
variable:
TC_BECOME).

-verbose

Specifies that you
want to receive a
confirmation message
after you issue this
command

In our case, the command syntax would look like this:

```
teamc Defect -design d960614  
-remarks 'board.hpp needs to be corrected'
```

9.3.3.3 Moving a Defect to the Size State

Use the **Size Defects** window (see Figure 134) to move a defect from the **design** state to the **size** state, so that the effort to resolve the defect can be sized.

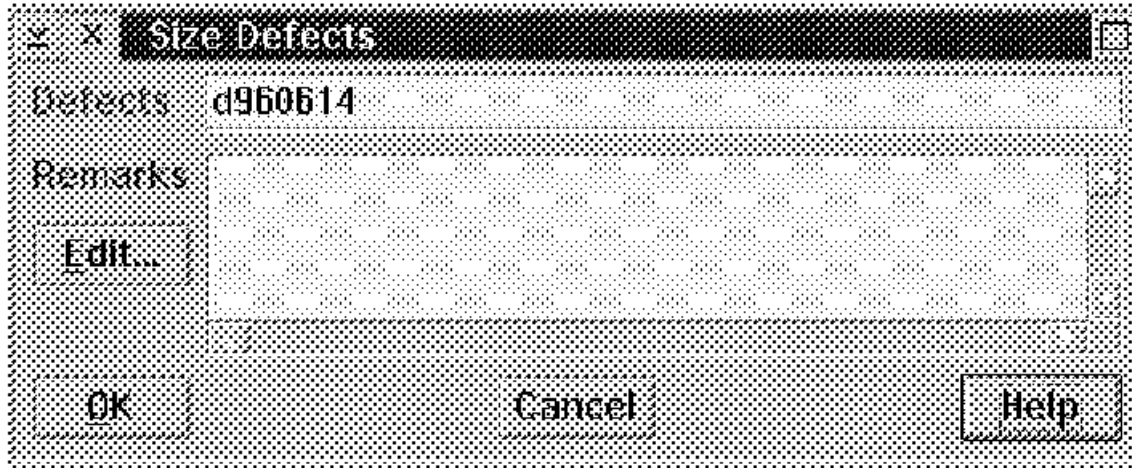


Figure 134. Size Defects Window

Command Line Interface Syntax

The following is the command line interface syntax for moving a defect to the **size** state:

```
teamc defect -size Name ... -family Name [-remarks Text]
                    [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-size	

Note: The family, remarks, become, and verbose attribute flags and their arguments are explained in "Moving a Defect to the Design State" in topic 9.3.3.2.

In our case, the command syntax would look like this:

```
teamc Defect -size d960614
```

When a defect is in the **size** state, you can create *sizing records*. You can create one *sizing record* per release and component pair. The defect owner has authority to create *sizing records*. By default, the owner of the sizing record is the owner of the component for which it was created. After all sizing records have been accepted or rejected, you can move the defect to the **review** state.

Use the **Create Sizing Records** window (see Figure 135) to create a sizing record for the defects or features you own. The defect or feature owner must create the sizing record. The sizing record indicates the time and resources needed to resolve a defect or implement a feature in one component for a release. A defect or feature name, a component, and a

release uniquely identify each sizing record. To create a sizing record, ensure that the process associated with the component includes the DSR subprocess.

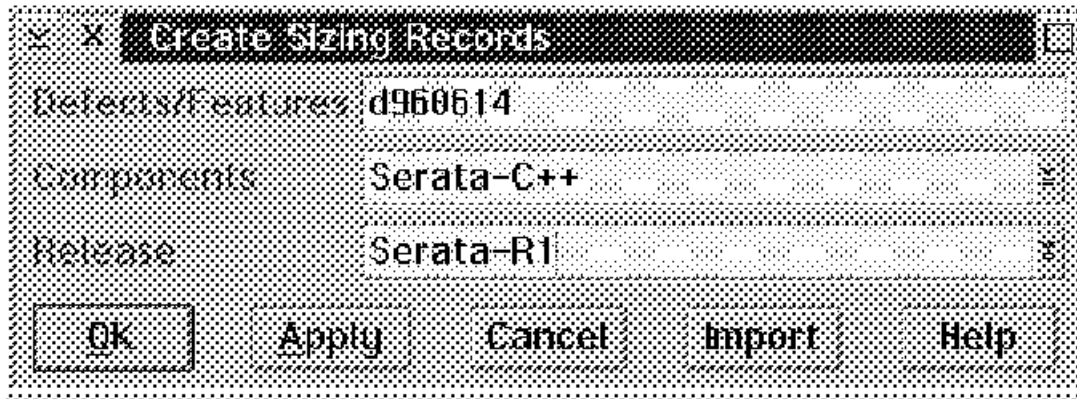


Figure 135. Create Sizing Records Window

Command Line Interface Syntax

The following is the command line interface syntax for creating a sizing record for the defect:

```
teamc size -create { -feature Name ... -defect Name ... }
                  -component Name ... -release Name
                  -family Name [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-create	Creates a sizing record for the corresponding defect or feature, release, and component Note: The associated component's process must include the DSR subprocess.
-feature <i>Name</i>	The feature identifiers for these sizing records
-defect <i>Name</i>	The defect identifiers for these sizing records
-component <i>Name</i>	The components for these sizing records (environment variable: TC_COMPONENT)
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	The family for which this command is being issued

TeamConnection at Large Moving a Defect to the Size State

```
(environment  
variable: TC_FAMILY)
```

Attribute flag and argument	Description
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Size -create -defect d960614 -component Serata-C++ -release Serata-R1
```

Use the **TeamConnection - Sizing Records** window (see Figure 136) to view sizing information for the defect or feature you specify. You get to the **Sizing Records** window by selecting the specific feature in the **TeamConnection - Features** window and then from the pop-up menu select **Show > Sizing records**. Sizing records indicate the time and resources needed to resolve one defect or to implement one feature in one component for one release.

The owner of the defect or feature must create the sizing records. You can create sizing records only when a defect or feature is in the **size** state. By default, the owner of a sizing record is the owner of the component to which the sizing record applies. After the sizing record owner accepts or rejects each sizing record, the owner can move the defect or feature from the **size** state to the **review** state.

TeamConnection automatically creates fix records for a defect or feature for each accepted sizing record when the defect or feature is accepted.

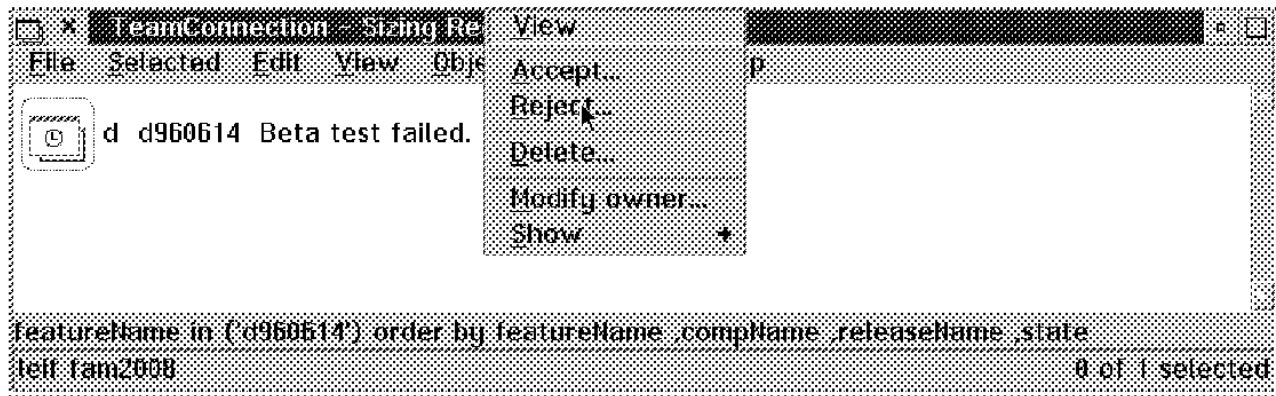


Figure 136. Sizing Records Window

Use the **Accept Sizing Records** window (see Figure 137) to size the effort involved in fixing the defect or implementing the feature. If you access the **Accept Sizing Records** window by selecting a defect or feature and using the selected pull-down or by using the pop-up menu (by selecting the defect or feature using mouse button 2), TeamConnection puts the correct data in all fields but the Sizing field.

Enter the sizing information and select **OK** to accept the sizing record. In our case we estimate the required time to be 0.5 man hours, and we have assigned *Felix* to do the job.

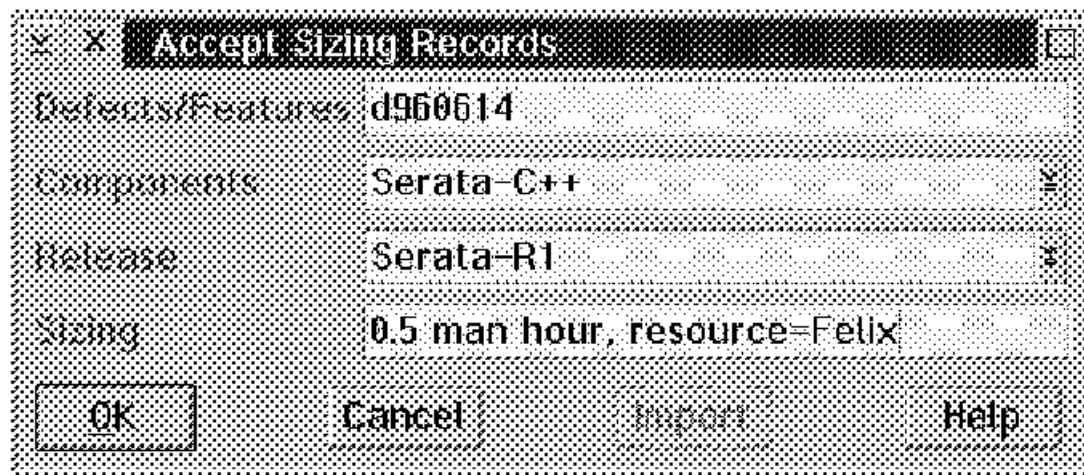


Figure 137. Accept Sizing Records Window

Command Line Interface Syntax

The following is the command line interface syntax for accepting the sizing record for the defect:

```
teamc size -accept { -feature Name ... -defect Name ... }
                  -component Name ... -release Name
                  -family Name -sizing Text [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-accept	Indicates that the sizing information is entered and complete for the corresponding defect or feature, release,

	and component. This action is used to record initial sizing information.
-feature <i>Name</i>	The feature identifiers for these sizing records
-defect <i>Name</i>	The defect identifiers for these sizing records
-component <i>Name</i>	The components for these sizing records (environment variable: TC_COMPONENT)
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	The family for which this command is being issued (environment variable: TC_FAMILY)
-sizing <i>Text</i>	The sizing information for the proposed defect or feature change in the specified component and release

Attribute Flag and Argument	Description
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Size -accept -defect d960614 -component Serata-C++  
-release Serata-R1 -sizing "0.5 man hour, resource=felix"
```

When all sizing records for a defect or feature have been either accepted or rejected, you can move the defect or feature to the **review** state. Fix records are created for all sizing records marked accept after the defect or feature is accepted.

After having accepted the *sizing record*, we can now move the feature to the **review** state.

9.3.3.4 Moving a Defect to the Review State

Use the **Review Defects** window (see Figure 119 in topic 9.2.2.4) to move a defect from the **size** state to the **review** state so that a formal review of the proposed defect resolution and its sizing information can occur. All sizing records for the defect must be accepted or rejected.

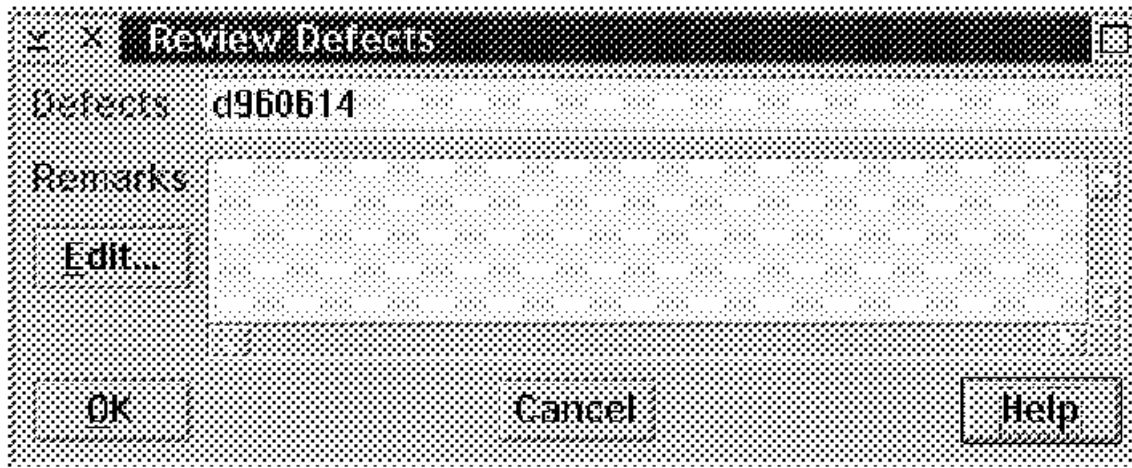


Figure 138. Review Defects Window

Command Line Interface Syntax

The following is the command line interface syntax for moving a defect to the **review** state:

```
teamc defect -review Name ... -family Name [-remarks Text]
                [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-review Name ...	Moves defects from the size state to the review state so that the proposed design implementation and sizing information can be reviewed
Note: The family, remarks, become, and verbose attribute flags and their arguments are explained in "Moving a Defect to the Design State" in topic 9.3.3.2.	

In our case, the command syntax would look like this:

```
teamc Defect -review d960614
```

Defects or features move to the **review** state after they have been sized. In the **review** state, the design text and sizing records are reviewed to determine the feasibility of the proposal. The owner can do one of the following:

- Accept the defect or feature if all design and sizing records are acceptable. This moves the defect or feature to the **working** state.
- Return the defect or feature to the originator if all design and sizing records are not acceptable. If necessary, the originator can reopen a defect or feature.
- Move the defect or feature back to the **design** state if design modifications are needed.

In our case, we believe that our design is accurate, so we can move the defect to the **working** state. In the next section we show you how to move a defect to the **working** state.

9.3.3.5 Accepting the Defect

Use the **Accept Defects** window (see Figure 139) to move a defect from the **open** or **review** state to the **working** state. If the process for the component includes the DSR subprocess, the defect must be in the **review** state. If the process for the component does not include the DSR subprocess, the defect must be in the **open** or **returned** state.

A verification record is created in the *notReady* state. If the release includes the track subprocess, TeamConnection creates a work area for each sizing record for the defect after the defect is accepted.

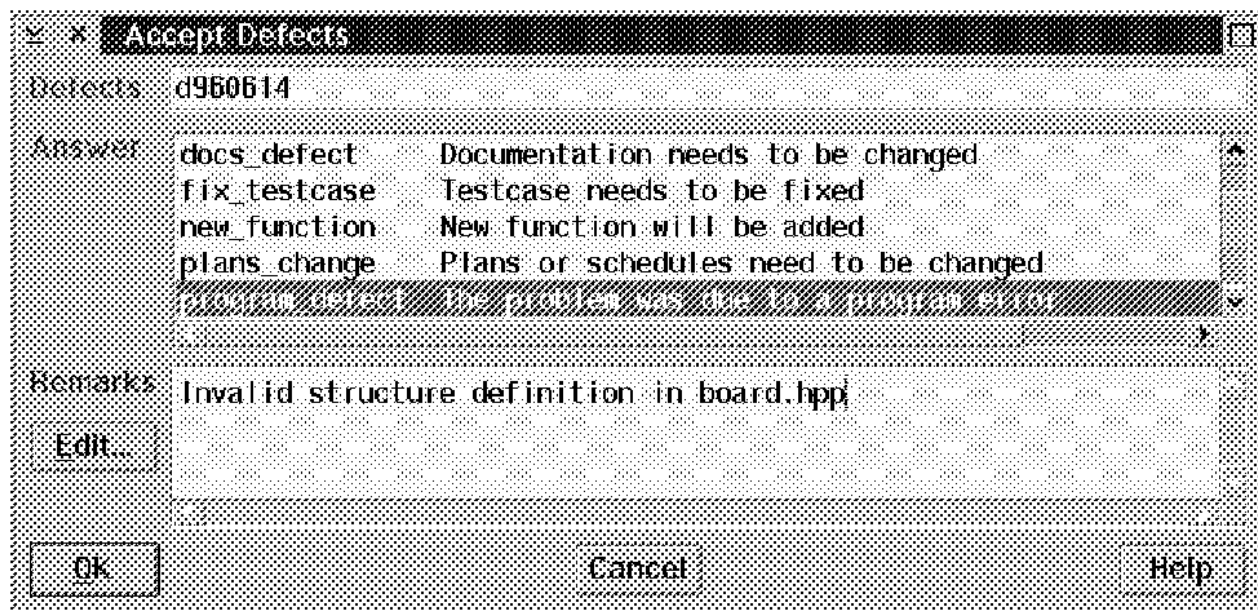


Figure 139. Accept Defects Window

Command Line Interface Syntax

The following is the command line interface syntax for accepting a defect and moving it to the **working** state:

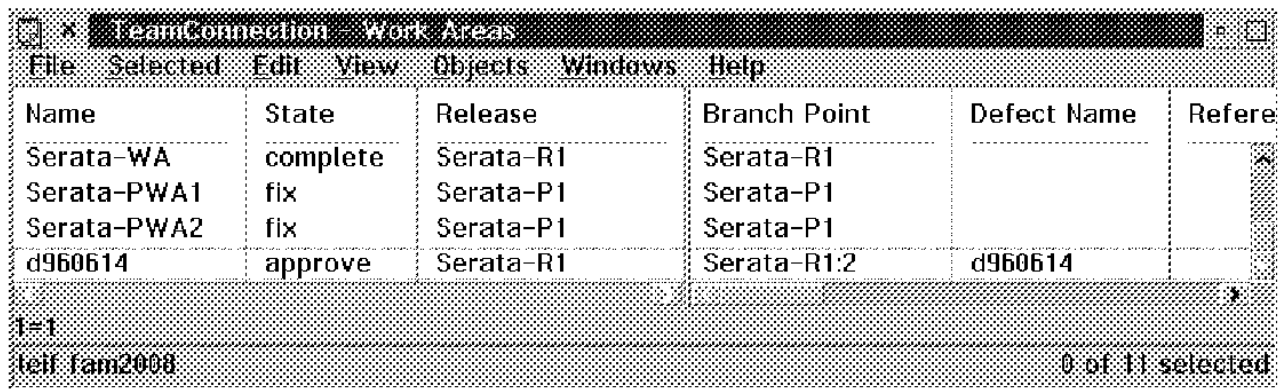
```
teamc defect -accept Name ... -family Name [-answer Name]*
          [-remarks Text] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-review Name ...	Moves defects from the size state to the review state so that the proposed design implementation and sizing information can be reviewed
-answer Name ...	Specifies an answer code when accepting, modifying, or returning a defect
Note: The family, remarks, become, and verbose attribute flags and their arguments are explained in "Moving a Defect to the Design State" in topic 9.3.3.2.	

In our case, the command syntax would look like this:

```
teamc Defect -accept d960614 -answer program_defect
-remarks 'Invalid structure definition in board.hpp'
```

When you accept a defect or feature, you accept the responsibility of resolving it. A defect or feature might require changes in more than one release. If the component includes the DSR subprocess, these releases were identified during the **size** state, and TeamConnection created a work area for each identified release (see Figure 140). If the component does not include the DSR subprocess, you will have to create a work area yourself.



Name	State	Release	Branch Point	Defect Name	Refere
Serata-WA	complete	Serata-R1	Serata-R1		
Serata-PWA1	fix	Serata-P1	Serata-P1		
Serata-PWA2	fix	Serata-P1	Serata-P1		
d960614	approve	Serata-R1	Serata-R1:2	d960614	

1=1
leif lam2008 0 of 11 selected

Figure 140. TeamConnection - Work Areas Window

9.3.3.6 The Defect in the Working State

Defects or features move to the **working** state when the owner accepts the defect or feature when it is in the:

- ☐ **Review** state if the DSR subprocess is included in the process
- ☐ **Open** state if the DSR subprocess is not included in the process

If any of the release subprocesses is defined for the release, a work area will be created (see Figure 140 in topic 9.3.3.5) for the defect or feature, and any changes to parts in the release can be done only through this defect or feature work area. Depending on which subprocess has been defined, specific actions have to be taken to move the work area from state to state.

In our case, we have the release *preship* process defined. Thus we have all of the release subprocesses defined, and the work area created will have to go through all of the different work area states.

9.3.3.7 The Work Area in the Approval State

When a work area is created, it goes to the **approve** state (see Figure 140 in topic 9.3.3.5) if the *approval subprocess* is included in the release process. TeamConnection creates an **approval record** for each user on the release's approver list. You can also create approval records manually for a work area without changing the approver list associated with the release. To create an *approval record*, the process associated with the release must include the *approval subprocess*.

The user assigned as the approver must approve any parts changed for the specified work area. An approver must accept or reject an approval record based on proposed part changes. The approver can also abstain from giving an opinion. When the owner of each approval record associated with a work area has accepted or abstained on the approval record, the work area moves automatically from the **approve** state to the **fix** state. If approval records have been rejected, the work area cannot move to the **fix** state, and part changes cannot be made.

Use the **TeamConnection - Approval Records** window (see Figure 141) to work with approval records. You accept, reject, or abstain the *approval record* from this window.

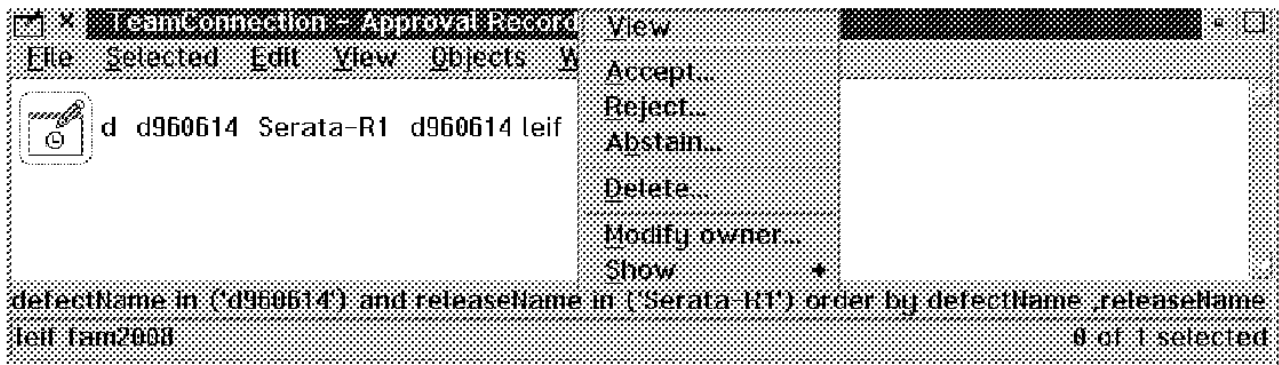


Figure 141. Approval Records Window

Use the **Accept Approval Records** window to complete an *approval record* by accepting it. When all approval records are in the accept or abstain state, the work area moves automatically to the **fix** state. If an approval record is in the reject state, the work area cannot move to the **fix** state.

The following fields are available in the **Accept Approval Records** window (see Figure 142):

Work areas

Type the names of the work areas whose approval records you want to accept.

Releases

Type the names of the releases associated with the work areas whose approval records you want to accept.

Approver

Type the user ID of the owner of the approval record.



Figure 142. Accept Approval Records Window

Command Line Interface Syntax

The following is the command line interface syntax for accepting an approval record for the defect:

```
teamc approval -accept -release Name ... -family Name
               -workarea Name ... [-approver Name]
               [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-accept	Approves the proposed part changes for the specified work area
-release <i>Name ...</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	The family for which this command is being issued (environment variable: TC_FAMILY)
-workarea <i>Name ...</i>	The work area associated with the approval record (environment variable: TC_WORKAREA)
-approver <i>Name</i>	The user ID of the owner of the approval record
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Approval -accept -workarea d960614 -release Serata-R1
               -approver leif
```

TeamConnection at Large
The Work Area in the Approval State

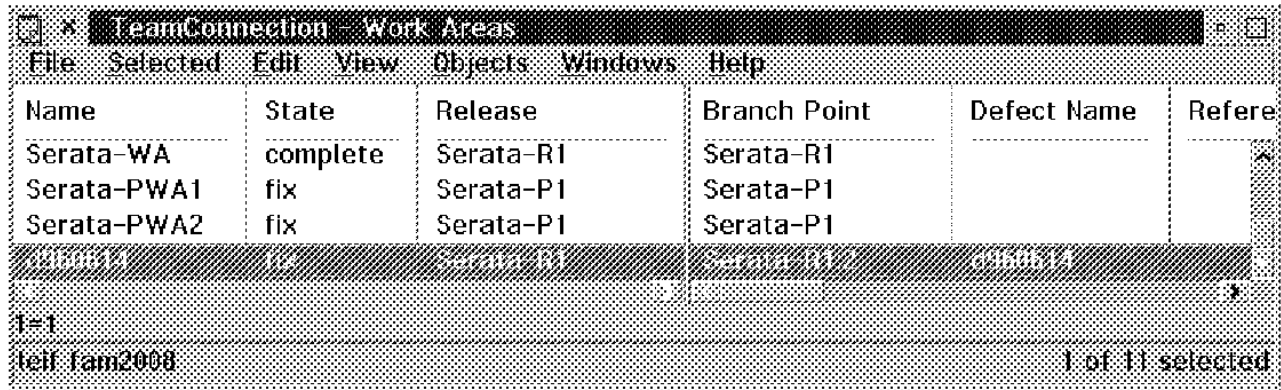
When the *approval record* has been accepted, you might notice that the icon for the *approval record* in the **TeamConnection - Approval Records** window (see Figure 143) has changed to PICTURE 137 .



Figure 143. Approval Records Window after Accept

9.3.3.8 The Work Area in the Fix State

After the *approval records* have been accepted, the **TeamConnection - Work Areas** window (see Figure 144) shows that the defect work area has changed to the **fix** state. Thus you can now start to work on the parts related to this defect. You can check out, modify, check in, build, extract, and test the part(s) affected by the defect.



Name	State	Release	Branch Point	Defect Name	Refere
Serata-WA	complete	Serata-R1	Serata-R1		
Serata-PWA1	fix	Serata-P1	Serata-P1		
Serata-PWA2	fix	Serata-P1	Serata-P1		

1 of 1 selected

Figure 144. TeamConnection - Work Areas Window

When the work is done and you are satisfied with the result, you will (depending on the defined release subprocesses) either integrate the work area or add it as a **driver member**. As we have all release subprocesses defined, including the *driver subprocess*, we will have to add it as a driver member to a driver.

9.3.3.9 Integrating the Work Area

Before you add defect or feature work areas as driver members to a driver, you have to *complete* the *fix records* associated with the work areas.

Use the **Complete Fix Records** window (see Figure 145) to move a fix record to the **complete** state. The **complete** state indicates that the part changes necessary to fix the defect or feature within that component are completed and checked in. When all fix records for a work area have been completed, the work area automatically moves to the **integrate** state.

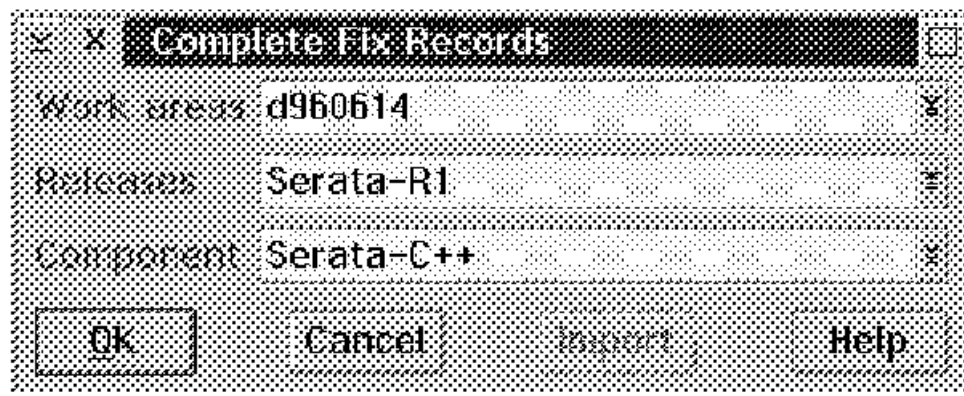


Figure 145. Complete Fix Records Window

Command Line Interface Syntax

The following is the command line interface syntax for completing the fix records for the defect:

```
teamc fix -complete -workarea Name ... -family Name
          -release Name ... -component Name [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-complete	Moves a fix record to the complete state to indicate that all part changes required in the associated component are completed. If no other fix records exist or all other records are completed, the work area changes from the fix state to the next valid state, which is governed by the release's process.
-workarea Name ...	The work area associated with the approval record (environment variable: TC_WORKAREA)
-family Name	The family for which this command is being issued (environment variable: TC_FAMILY)

Attribute Flag and Argument	Description
-release <i>Name ...</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-component <i>Name</i>	Specifies the component that manages the parts to be changed (environment variable: TC_COMPONENT)
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Fix -complete -workarea d960614 -release Serata-R1
-component Serata-C++
```

If you are using the *track* subprocess, use the **Integrate Work Areas** window (see Figure 146) to move a work area from the **fix** state to the **integrate** state when you finish making all of the part changes. If you are not using the *track* subprocess, the work area automatically moves from the **fix** state to the **complete** state.

A work area moves to the **integrate** state automatically when all of the fix records for the work area move to the **complete** state. You can force a work area to the **integrate** state if you do not have to make part changes. For example, if a change for another defect or feature fixes the current defect or feature, you do not have to make part changes for this work area. Any fix records that exist must be in the **complete** state.

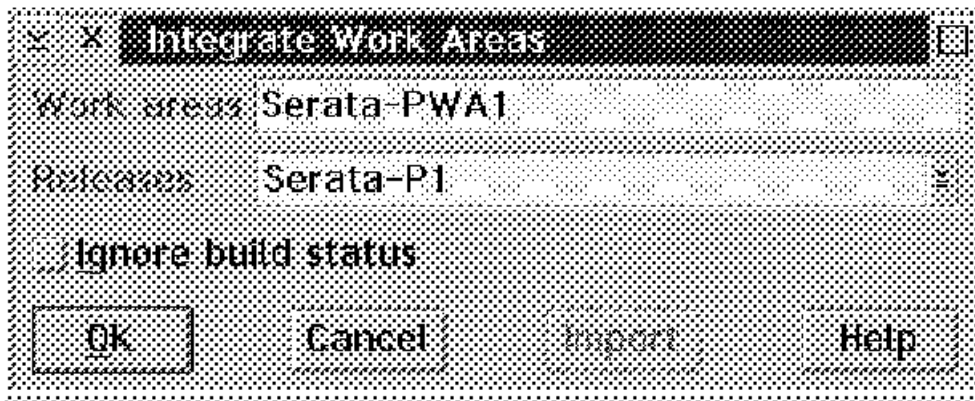


Figure 146. Integrate Work Areas Window

Command Line Interface Syntax

The following is the command line interface syntax for integrating a work area:

```
teamc workarea -integrate Name ... -release Name ...
               -family Name [-force] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-integrate <i>Name ...</i>	Changes the state of the specified work areas from fix to the next valid state governed by the release's process. For a release whose process includes the driver subprocess, this action is valid only if part changes were not made for the work area and the work area is not committed in a driver.
-release <i>Name ...</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	The family for which this command is being issued (environment variable: TC_FAMILY)
-force	Continues with the operation even if build output in the work area is out-of-date

Attribute	Description
-----------	-------------

flag and argument	
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

Even if we do not do any specific *integrate* for our defect work area, TeamConnection still does an integrate. When we complete all of the *fix records* for our particular work area, TeamConnection automatically integrates the work area for us. The following shows the command as it appears in the *teamc.log* file:

```
teamc WorkArea -integrate d960614 -release Serata-R1
```

9.3.3.10 Creating a Driver and Adding a Driver Member

Before you can add any defect or feature work areas as driver members, you have to create the driver. As you might remember from "Actions They Can Perform" in topic 9.1.2, a user with the authority group of *projectlead* is the only user allowed to create a *driver* (except for the *superuser*).

A driver is comprised of a number of work areas. Because a work area is a mechanism for grouping changes to parts (those that are modified for a particular defect or feature), a driver is a mechanism for grouping multiple sets of changes to parts in a release.

TeamConnection drivers provide snapshots of a release at different points in the release's development life cycle. Each committed driver can be re-created at a later date by extracting a tree view represented by the driver. To create a driver, you assign a name to it and relate it to a release. You can then create driver members from your work areas. These driver members represent the changes to the release. If you create a driver, you are the driver owner; however, you can reassign ownership of the driver to another user.

When you want to make permanent all part changes associated with the driver, you can move the driver to the **commit** state. However, all member work areas must be in the **integrate** state or the **commit** state, all prerequisite work areas must be included in the driver, and all *collision* records associated with the driver must be either accepted, rejected, or reconciled.

You can indicate when a driver is ready for formal testing by moving the driver into the **complete** state. This action changes the state of the associated work areas to the **test** state if the release has an environment list; otherwise, the work areas move to the **complete** state.

You can extract a driver at any time after you add work areas to the driver. You can extract the driver in one of the following forms:

Delta part tree

A directory structure that contains only the driver members for the driver. If the driver is in the **integrate** state, you can only extract a delta part tree.

Full part tree

A directory structure that contains all of the driver members, regardless of whether they were changed, for the committed driver

Extracting the full part tree re-creates the release as it existed at the time that the driver was committed. Extracting the delta part tree re-creates only the parts that were changed for that driver.

Use the **Create Drivers** window (see Figure 147) to create a new driver to group part changes within the release. When you create a driver, you are the driver owner. The release process includes the driver subprocess and track subprocess.

The following fields are available in the **Create Drivers** window:

Drivers

Type the names of the drivers that include the parts that changed.

Release

Type the name of the release to which the changed parts belong.

Type

Select the type of driver that you are assigning to the driver.

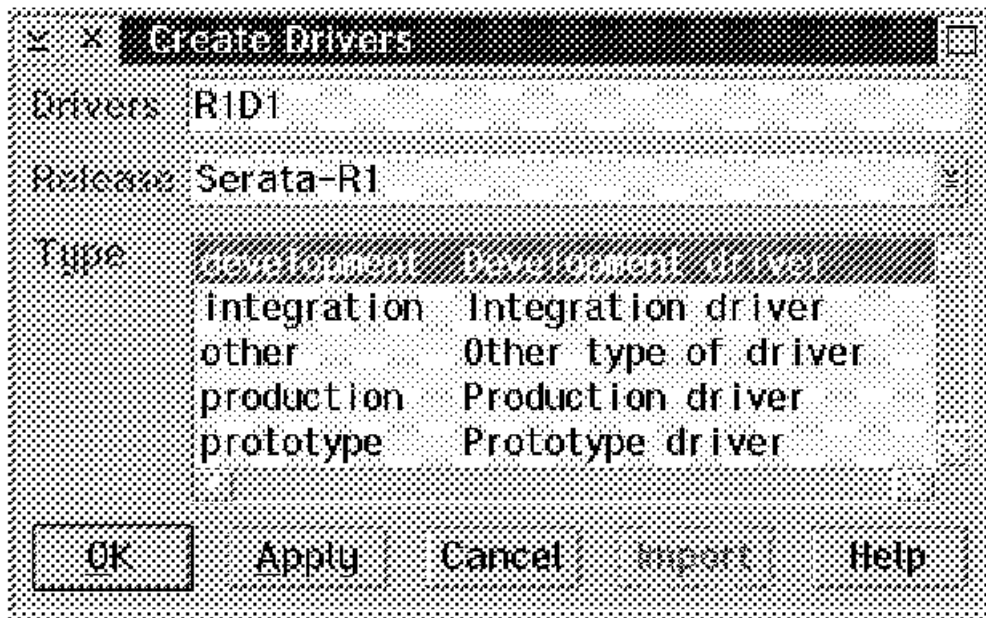


Figure 147. Create Drivers Window

Command Line Interface Syntax

The following is the command line interface syntax for creating a driver:

```
teamc driver -create Name ... -release Name -family Name [-type Name]
               [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-create <i>Name</i>	Creates drivers with the specified names. The user who creates a driver is the driver owner by default.
-release <i>Name ...</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	The family for which this command is being issued (environment variable: TC_FAMILY)
-type <i>Name ...</i>	Specifies the type of driver when creating or modifying a driver. You can use the report command to find out the types of drivers.
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from

TeamConnection at Large Creating a Driver and Adding a Driver Member

your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).

-verbose

Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Driver -create R1D1 -release Serata-R1 -type development
```

Once the driver has been created, it will be in the **working** state (see Figure 148), and it will stay in the **working** state until the first work area is added as a driver member.

Driver	Release	State	Type	Owner	Area	Created
R1D1	Serata-R1	working	development	leif	BLD80/E2_RM412	1996/06/17 1

1=1
leif lam2008 0 of 1 selected

Figure 148. TeamConnection - Drivers Window

Use the **Add Driver Members** (see Figure 149) window to add new driver members to a driver. These driver members designate your work areas as members of a specific driver. To add driver members, the process of the associated release must include the driver subprocess. A driver automatically moves to the **integrate** state when you add the first work area as a driver member.

The following fields are available in the **Add Driver Members** window:

Driver

Type the name of the driver to which specified work areas are added.

Release

Type the name of an existing release that contains the driver to which you want to add new driver members.

Work areas

Type the names of the defects or features being monitored by the work areas you want to add as driver members. You can import selected defects or features from the **Work Areas** window.

Ignore build status

Select the Ignore build status check box to ignore the build status of the parts and force the addition of the driver members.

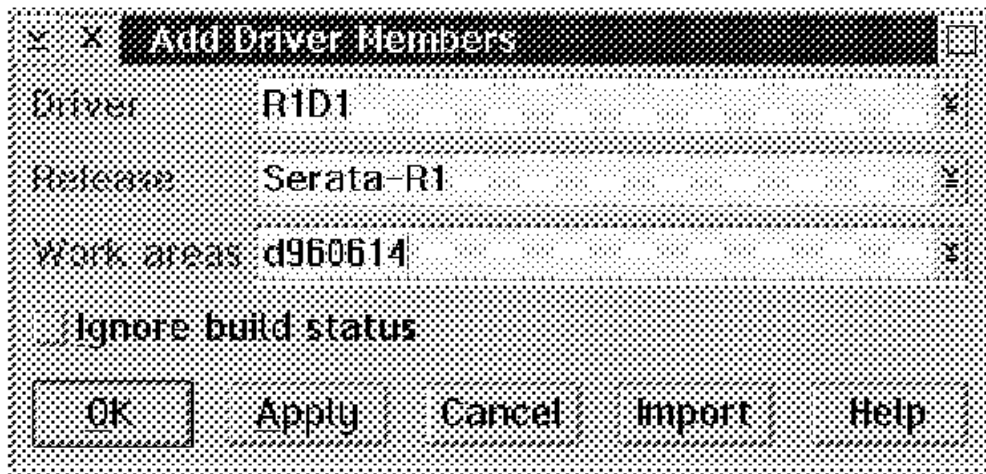


Figure 149. Add Driver Members Window

Command Line Interface Syntax

The following is the command line interface syntax for adding a driver member:

```
teamc driverMember -create -driver Name -release Name -family Name
                    -workarea ... [-force] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-create <i>Name</i>	Creates work areas as members of a specific driver Note: To perform this action, the associated release's process must include the driver subprocess.
-release <i>Name ...</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	The family for which this command is being issued (environment variable: TC_FAMILY)
-workarea <i>Name ...</i>	Specifies the work area or work areas you want to include in or remove from a driver (environment variable: TC_WORKAREA)
-force	Continues with the operation even if build output is out-of-date.

TeamConnection at Large

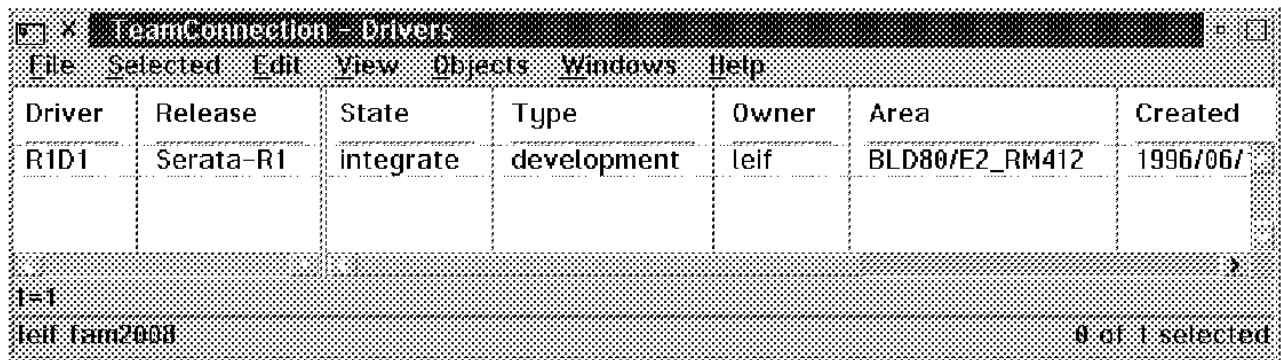
Creating a Driver and Adding a Driver Member

-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc DriverMember -create -driver R1D1 -release Serata-R1
-workarea d960614
```

As soon as the first work area is added to the driver as a member of the driver, the driver automatically moves to the **integrate** state (see Figure 150).



TeamConnection - Drivers						
Driver	Release	State	Type	Owner	Area	Created
R1D1	Serata-R1	integrate	development	leif	BLD80/E2_RM412	1996/06/
1=1						
leif lam2008 0 of 1 selected						

Figure 150. TeamConnection - Drivers Window after Driver Has Moved to Integrate State

Adding a work area as a driver member includes the part changes monitored by that work area in the driver. You can add a work area in the **integrate** state as a driver member; however, a driver cannot be committed unless all of its member work areas are in the **integrate** state. Thus, a driver can include a partial fix for testing purposes without allowing a partial fix to be committed in the driver.

After you define the new driver, check for any existing prerequisite or corequisite work area relationships. If you find any, either remove some of the driver members or add work areas to the driver to resolve the prerequisite and corequisite requirements.

When no prerequisites or corequisites remain, you can extract the parts in preparation for compilation. When a driver is in the **integrate** state, you can only extract a delta part tree. This extract action copies all of the parts associated with the work areas that are members of the driver.

After you complete integration testing of a new driver, the release can be updated by committing the driver.

9.3.3.11 Committing the Driver

Committing a driver commits all work areas that were designated as driver members and all part changes included in those work areas. When you commit the changes in a driver, they establish a new baseline for subsequent development of the release.

You commit a driver to finalize all of the part changes included in the driver. If any outstanding prerequisite or corequisite work area relationships exist for the driver, you cannot commit the driver. When you commit a driver, all work areas that are driver members change from the **integrate** state to the **commit** state, and the part changes represented by those work areas become permanent.

The only changes you can make to a committed driver are changing its owner and changing its type. If you discover a defect in a committed driver, you must open a new defect and make further changes to parts in a new driver.

You can extract the committed driver to produce a new base part tree against which the next set of changes will be applied. You can extract the full part tree at any time for any previously committed driver.

A single work area can be a member of more than one driver. If you commit one of the drivers including this work area, the state of the work area changes to **commit**. Other drivers that include this work area disregard the work area.

Use the **Commit Drivers** window (see Figure 151) to move a driver to the **commit** state so that it can no longer be modified.

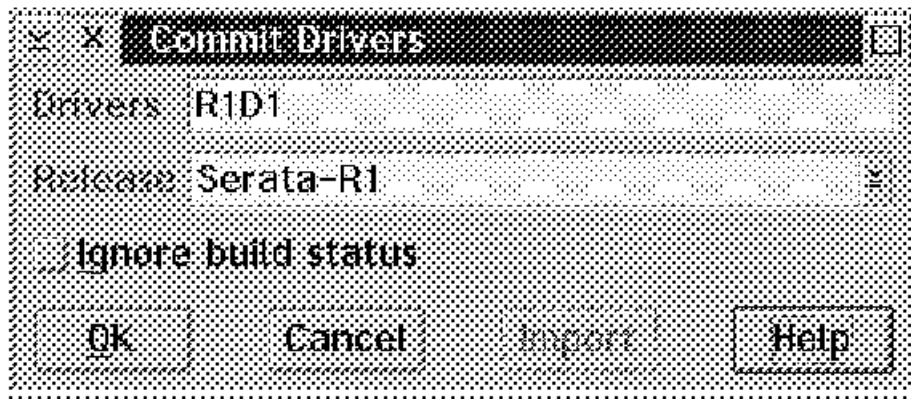


Figure 151. Commit Drivers Window

Command Line Interface Syntax

The following is the command line interface syntax for committing a driver:

```
teamc driver -commit Name ... -release Name
          -family Name [-force] [-become Name]
          [-verbose]
```

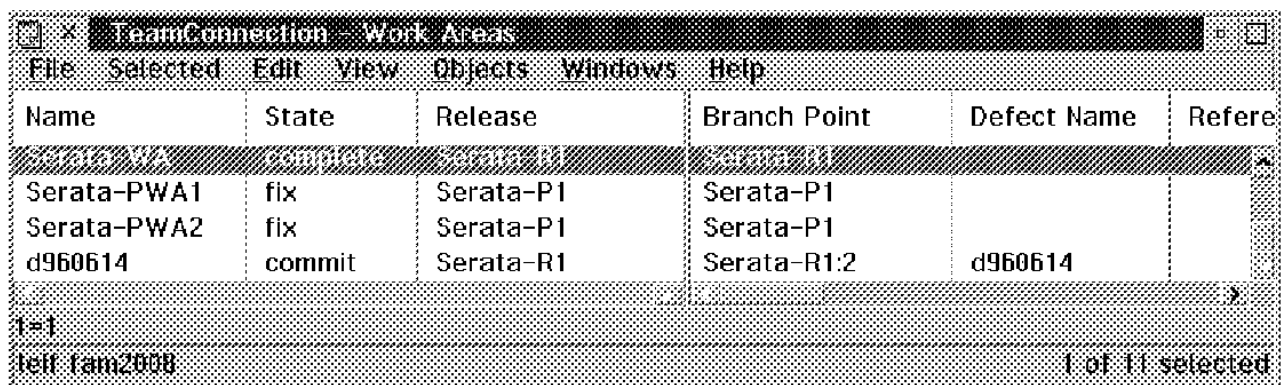
Attribute Flag and Argument	Description
-commit Name ...	Moves the specified drivers to the commit state where they can no longer be modified. All part changes associated with driver members become permanent.
-release	The release for

Name ...	which this command is being issued (environment variable: TC_RELEASE)
-family Name	The family for which this command is being issued (environment variable: TC_FAMILY)
-force	Continues with the operation even if build output is out-of-date
-become Name	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Driver -commit R1D1 -release Serata-R1
```

When a driver is committed, you can extract a full part tree that includes all of the parts in the associated release. Work areas that are members of a committed driver automatically move to the **commit** state (see Figure 152).



Name	State	Release	Branch Point	Defect Name	Refere
Serata-PWA1	fix	Serata-P1	Serata-P1		
Serata-PWA2	fix	Serata-P1	Serata-P1		
d960614	commit	Serata-R1	Serata-R1:2	d960614	

1=1
left ram2008 1 of 11 selected

Figure 152. TeamConnection - Work Areas Window after Driver Has Been Committed

Parts can no longer be changed with reference to work areas that are members of a committed driver. You can extract parts from a committed driver, even if there are more current versions of the same parts.

9.3.3.12 Completing the Driver

The final step in the driver subprocess is to complete the committed driver. Completing a driver activates the formal test subprocess by changing all work areas in the driver from the **commit** state to the **test** state. Complete committed drivers after the updated release has been distributed to the appropriate testers.

When a driver is complete, the part changes associated with the work areas that are driver members are ready for formal testing. Nothing more can be done to a driver in the **complete** state.

Use the **Complete Drivers** window (Figure 153) to move a driver to the **complete** state so that it is ready for formal testing.

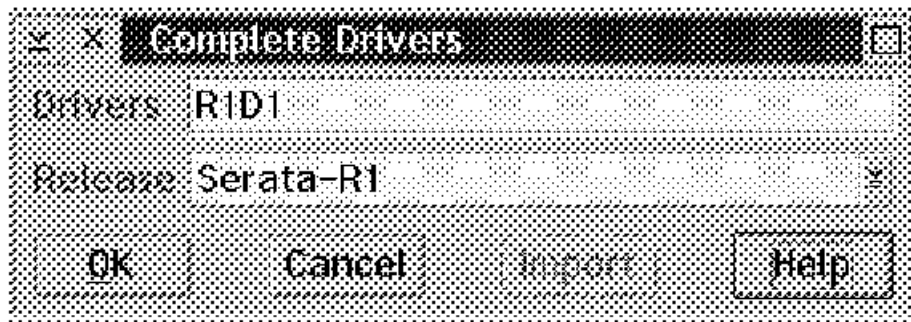


Figure 153. Complete Drivers Window

Command Line Interface Syntax

The following is the command line interface syntax for completing a driver:

```
teamc driver -complete Name ... -release Name -family Name
          [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-complete Name ...	Moves the specified drivers to the complete state where they are ready to be tested. All driver members change to the test or complete state.
-release Name ...	The release for which this command is being issued (environment variable: TC_RELEASE)
-family Name	The family for which this command is being issued (environment variable: TC_FAMILY)
-become Name	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access

	authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Driver -complete R1D1 -release Serata-R1
```

Work areas that are members of a completed driver automatically move to the **test** state (see Figure 154) if its release includes the test subprocess and the release has an environment list. Work areas that are members of a completed driver automatically move to the **complete** state if its release does not include the test subprocess or the release does not have an environment list.

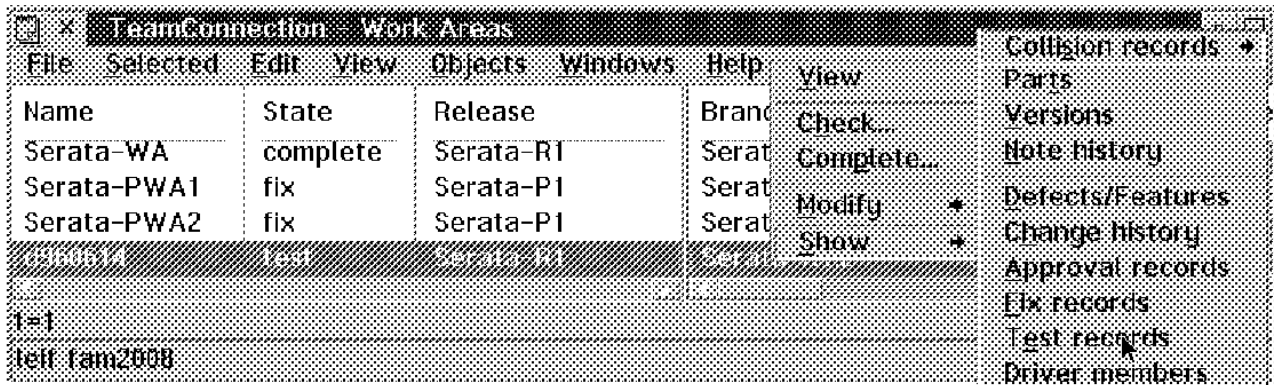


Figure 154. TeamConnection - Work Areas Window after Driver Has Been Completed

9.3.3.13 Completing Test Records

For test records to exist, the process for a release must include the *test subprocess*. The release must have an environment list. An environment list associates a tester with each environment listed for the release.

Whenever you create a new work area for a release, TeamConnection creates a test record for each environment in the environment list if the release has an environment list and if it includes the test subprocess. Each test record includes the environment name and user ID specified on the release environment list. The tester is the owner of the user ID and owns the test record.

Test records are activated (moved to the **ready** state, see Figure 155) when the associated work area moves to the **test** state.

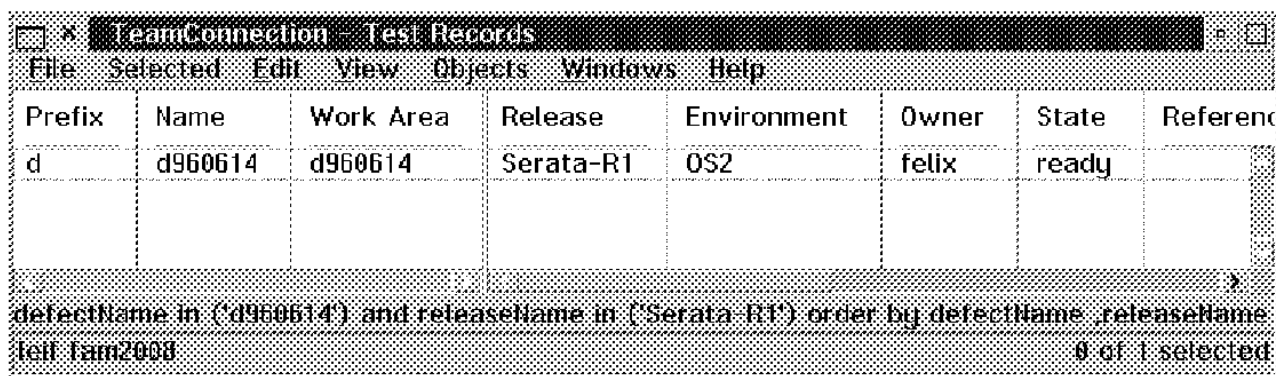


Figure 155. TeamConnection - Test Records Window

Use the **Accept Test Records** window (Figure 156) results for a release environment test by moving the test record to the complete state.

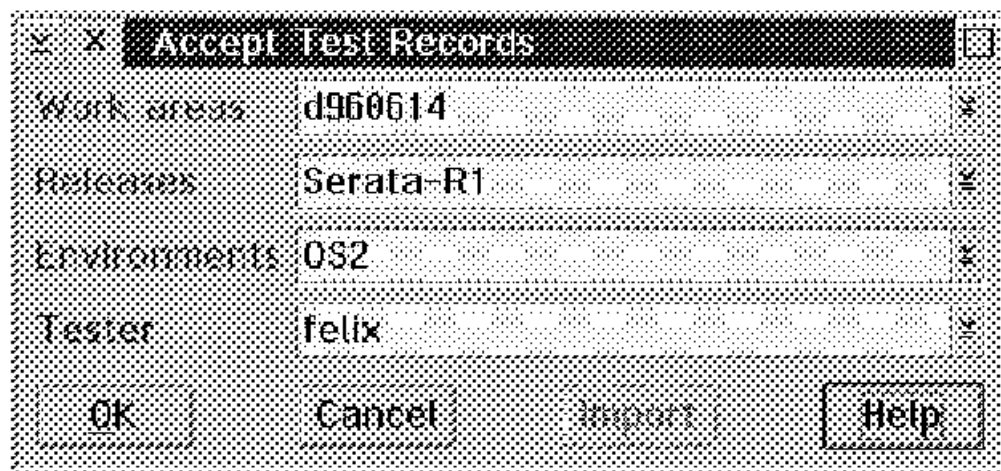


Figure 156. Accept Test Records Window

Command Line Interface Syntax

The following is the command line interface syntax for accepting a test record:

```
teamc test -accept -workarea ... -family Name
          -release Name ... -environment Name ...
          [-tester Name] [-become Name] [-verbose]
```

Attribute	Description
-----------	-------------

Flag and Argument	
-accept	Indicates successful results for a release environment test
-workarea ...	Specifies the work areas associated with the test records
-family <i>Name</i>	The family for which this command is being issued (environment variable: TC_FAMILY)
-release <i>Name</i> ...	The release for which this command is being issued (environment variable: TC_RELEASE)
-environment <i>Name</i> ...	Specifies the environments in which testing must be done
-tester <i>Name</i>	Performs test actions on a test record owned by a different user. By naming the tester, this flag identifies the test record to be modified.

Attribute flag and argument	Description
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Test -accept -workarea d960614 -release Serata-R1
          -environment OS2 -tester felix
```

When all test records associated with a work area have been accepted, rejected, or abstained, the work area automatically changes from the **test** state to the **complete** state (see Figure 157). If a test record has been rejected, you must open another defect or feature to address the changes that are still required.

TeamConnection - Work Areas					
File Selected Edit View Objects Windows Help					
Name	State	Release	Branch Point	Defect Name	Refere
Serata-WA	complete	Serata-R1	Serata-R1		
Serata-PWA1	fix	Serata-P1	Serata-P1		
Serata-PWA2	fix	Serata-P1	Serata-P1		
d960614	complete	Serata-R1	Serata-R1:2	d960614	
1=1					
leif lam2008				1 of 1 selected	

Figure 157. TeamConnection - Work Areas Window after All Test Records Have Been Completed

When the first work area moves to the **complete** state, the defect or feature automatically moves to the **verify** state (see Figure 158) or **closed** state. However, if a release is specified when the defect is created, the defect moves to the **verify** state when all work areas (there can be multiple work areas for the defect) for the release are integrated.

TeamConnection - Defects								
File Selected Edit View Objects Windows Help								
Prefix	Name	Component	State	Originator	Owner	Severity	Age	Refere
d	d960614	Serata-C++	verify	leif	leif	2	0	
1=1								
leif lam2008							0 of 1 selected	

Figure 158. TeamConnection - Defects Window after the First Work Area Has Moved to the Complete State

9.3.3.14 Completing the Verification Records

If the verify subprocess is configured for a component and the track subprocess is configured for the release specified for a defect, TeamConnection automatically moves the defect from the **working** to the **verify** state when its work area moves to the **complete** state. If the track subprocess is configured but no release is specified for a defect, TeamConnection automatically moves a defect or feature from the **working** to the **verify** state when the first work area associated with the defect or feature moves to the **complete** state. If the track subprocess is not configured, the owner of a defect or feature can select **Verify** for the specific defect or feature to move it from the **working** to the **verify** state.

If the verify subprocess is configured for the component to which a defect or feature is assigned, the originator of the defect or feature must certify that the defect or feature is satisfactorily resolved by completing a *verification record*. The originator must complete additional verification records if any defects or features are returned as duplicates of the defect or feature. If the verify subprocess is not configured, TeamConnection automatically moves the defect or feature to the **closed** state.

Use the **Accept Verification Records** window (see Figure 159) to complete a verification record by approving it. TeamConnection creates a verification record for the originator of a defect or feature when the defect or feature is accepted, if the component to which the defect or feature is assigned includes the verify subprocess.

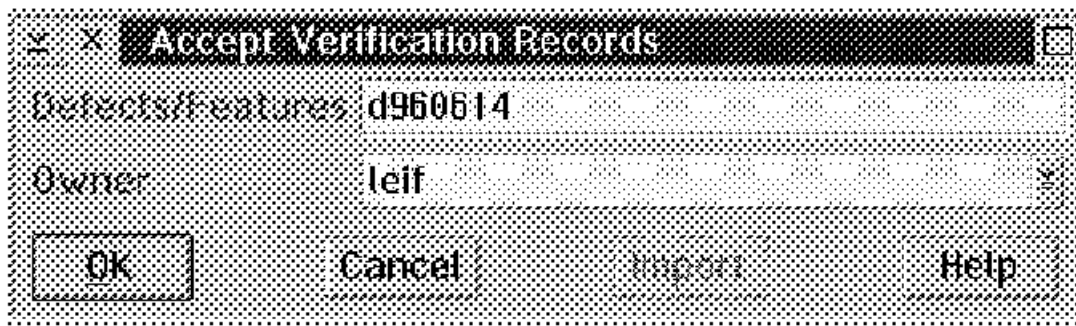


Figure 159. Accept Verification Records Window

Command Line Interface Syntax

The following is the command line interface syntax for accepting a verification record:

```
teamc verify -accept { -defect Name ... -feature Name ... }
                  [-family Name [-tester Name] [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-accept	Indicates successful verification of the defect resolution or feature implementation
-defect Name ...	Specifies the defect identifiers associated with the verification records
-feature Name ...	Specifies the feature identifiers associated with the verification records

-family Name	The family for which this command is being issued (environment variable: TC_FAMILY)
-tester Name	Identifies the owner of the verification record if you are performing the verification for someone else
-become Name	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc Verify -accept -defect d960614 -tester leif
```

When all verification records for a defect or feature have been either accepted, rejected, or abstained, and all of the work areas for that defect or feature are complete, the defect or feature automatically changes from the **verify** state to the **closed** state.

If the verify subprocess is not configured, TeamConnection automatically moves the defect or feature to the **closed** state. If you reject a verification record because you are not satisfied with the fix, you must open a new defect to address the problem. Rejecting the verification record does not prevent the defect or feature from closing.

Figure 160 shows the **TeamConnection - Defects, Work Areas, and Drivers** window after the defect has been closed.

TeamConnection - Defects								
Prefix	Name	Component	State	Originator	Owner	Severity	Age	Ref
d	d960614	Serata-C++	closed	leif	leif	2	0	

TeamConnection - Work Areas					
Name	State	Release	Branch Point	Defect Name	Refere
Serata-WA	complete	Serata-R1	Serata-R1		
Serata-PWA1	fix	Serata-P1	Serata-P1		
Serata-PWA2	fix	Serata-P1	Serata-P1		
d960614	complete	Serata-R1	Serata-R1:2	d960614	

Figure 160. TeamConnection - Defects, Work Areas, and Drivers Windows
after Defect Has Been Closed

9.4 Summary

TeamConnection subprocesses govern the state changes for TeamConnection objects. The *design*, *size*, and *review* (DSR) and *verify* subprocesses are configured for component processes. The *track*, *approve*, *fix*, *driver*, and *test* subprocesses are configured for release processes. By combining the subprocesses you can have either relatively loose control or very stringent control of your development project depending on where in the development life cycle your project is.

TeamConnection is shipped with a basic set of authority groups for typical users in a software development environment. An authority group is a list of allowable actions. You will belong to a certain authority group depending on your role in the project. Your family administrator can modify the authority groups or create new groups to reflect the needs of your organization.

TeamConnection uses the following records to control the movement of defects, work areas, and drivers:

Approval record

A status record that an approver must accept or reject based on proposed part changes. The approver can also abstain from giving an opinion.

Fix record

A status record that is associated with a work area

Sizing record

A status record created for each component and its associated release that indicates the approximate amount of work needed to resolve the defect or implement the feature

Test record

A status record used to record the outcome of an environment test

Verification record

A status record that the originator of a defect or a feature must mark before the defect or feature can move to the closed state

The various records are associated with lists of users who have to either accept, reject, or abstain the specific record. Involving several people in the approval of a proposed change minimizes the risk of introducing that specific change. For example, a new change could affect the way a certain function or class method is invoked and by introducing the new change the function or class method can not be invoked anymore.

10.0 Chapter 10. Advanced Topics

In this chapter we take a look at some of the more advanced functions of TeamConnection. We cover these topics:

- ☐ Concurrent development (parallel development)
- ☐ Using configurable fields
- ☐ Using report facilities
- ☐ User exits

Subtopics

- 10.1 Concurrent Development
- 10.2 Using Configurable Fields
- 10.3 Using Report Facilities
- 10.4 User Exits

10.1 Concurrent Development

TeamConnection enables you to work with parts in a release in parallel, or **concurrent** (as it is called in TeamConnection) development mode. Thus multiple users can work on the same version of the same part at the same time. This is in contrast to serial mode, where only one user can work on the same version of the same part at the same time.

You turn on concurrent development mode for a release when you create the release. You can then check out parts to your work area. From your work area, you can modify and compile parts without the parts being visible to other users of the release. You make the parts visible to other users of the release by integrating (committing) your work area.

If any other users of the parts in the release have performed an integrate on the same version of the same parts on which you are working, and you then refresh or integrate the parts in your work area, TeamConnection generates collision records. Collision records indicate that another user has changed the parts on which you are working. Using the collision records, you can determine how to reconcile the changes you have made to the parts with the changes the other user has made. TeamConnection provides you with a merge tool that enables you to selectively merge two text files into one.

Figure 161 shows a high-level example of concurrent development:

1. At point **T1** in time, two developers check out part **A** and modify it, creating **A'** and **A''**.
2. Developer 1 integrates the work area containing part **A'** at **T2**.
3. Developer 2 wants to integrate the work area containing part **A''** at **T3**, but TeamConnection does not allow it and signals a *collision* between **A'** and **A''**. Unless **A''** supersedes all changes of **A'**, there is a need to merge both modules and create **A'''**. TeamConnection provides the developer with a graphical tool for the merge (see "Using the Merge Tool" in topic 10.1.4).
4. Developer 2 is allowed to integrate the work area containing part **A'''** at **T4**, after he or she has reconciled the changes through collision records.

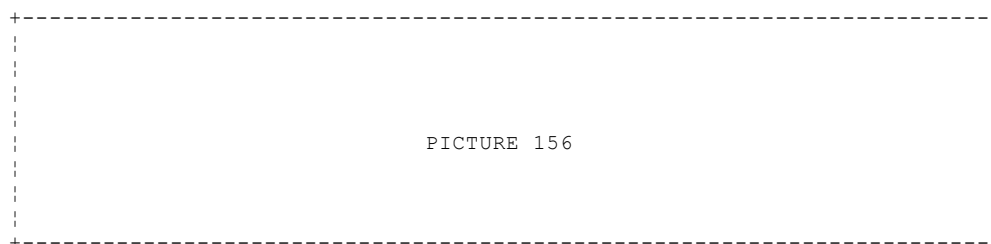


Figure 161. Concurrent Development

Even though the merge tool is valid only for text files (such as COBOL source code), you can still reconcile a collision between any two parts. Merging facilities are available only for coarse-grained parts such as files and not for fine-grained parts such as database designs and GUI objects. Thus concurrent development in its Release 1 implementation is probably best suited for 3GL environments. VisualGen, for example, when it detects a collision, gives the developer the choice of which object should be taken (**A'** or **A''**); in other words, VisualGen does not provide a merge facility.

In the sections that follow we discuss:

- ☐ Using concurrent development with no track process
- ☐ Using concurrent development with a track process
- ☐ Reconciling collision records
- ☐ Using the merge tool

Subtopics

- 10.1.1 Using Concurrent Development with No Tracking Process
- 10.1.2 Using Concurrent Development with a Tracking Process
- 10.1.3 Reconciling Collision Records
- 10.1.4 Using the Merge Tool

10.1.1.1 Using Concurrent Development with No Tracking Process

Figure 162 shows an action-state diagram for working with parts in a release that has concurrent development but does not follow a tracking process. In this example, a *collision* is detected at *refresh* time because earlier work area **B** was integrated with the release containing a change to the same part and version that were changed in work area **A**. In serial development, this cannot occur because the specific part would have been locked from other developers once it had been checked out.

In concurrent development, however, parts are not locked when they are checked out, so more than one developer can work on the same version of the same part. TeamConnection keeps track of potential collisions. If a collision occurs, TeamConnection creates a *collision record*. To *integrate* a work area containing parts for which *collision records* have been created, a reconciliation has to be performed. For text files, a reconciliation can be done through a merge by using the merge tool as shown in Figure 170 in topic 10.1.4. For nontext files, a reconciliation can be done only by selecting one of the parts as being the *new* part.

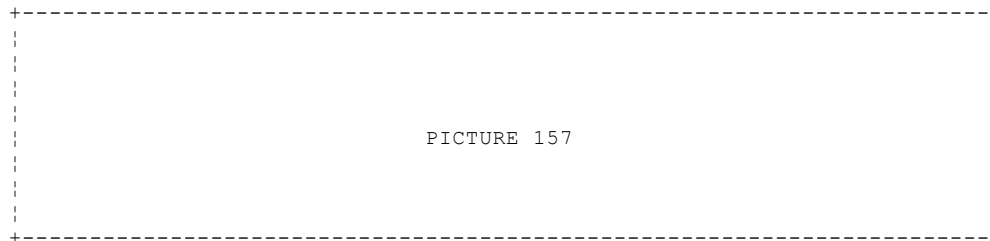


Figure 162. Concurrent Development with No Track Process

10.1.2 Using Concurrent Development with a Tracking Process

Figure 163 shows an action-state diagram for working with parts in a release that has concurrent development and follows a tracking process. Figure 163 starts at *add driver member* time, which means that the work area has already moved from the **fix** state to the **integrate** state. At *add driver member* time, a *collision* is detected and a *collision record* is created.

To reconcile, the work area for which the *collision record* was created has to be taken back to the **fix** state. To take the work area back to the **fix** state, a *fix record* has to be activated.

After the *collision records* have been reconciled, the *fix record* is completed, and the work area again moves to the **integrate** state. The work area whose collision records were reconciled can now be added as a driver member to the driver.

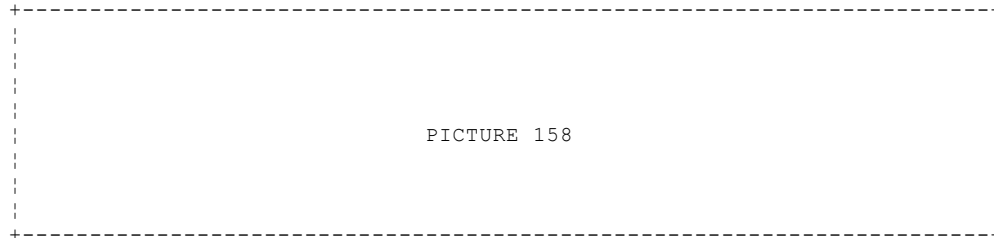


Figure 163. Concurrent Development with a Track Process

10.1.3 Reconciling Collision Records

In Figure 164 we show two windows representing two work areas: *Serata-PWA1* and *Serata-PWA2*. In *Serata-PWA1* the *serata.c* file is locked by user *felix*, and in *Serata-PWA2* the same version of the same file is locked by user *leif*.

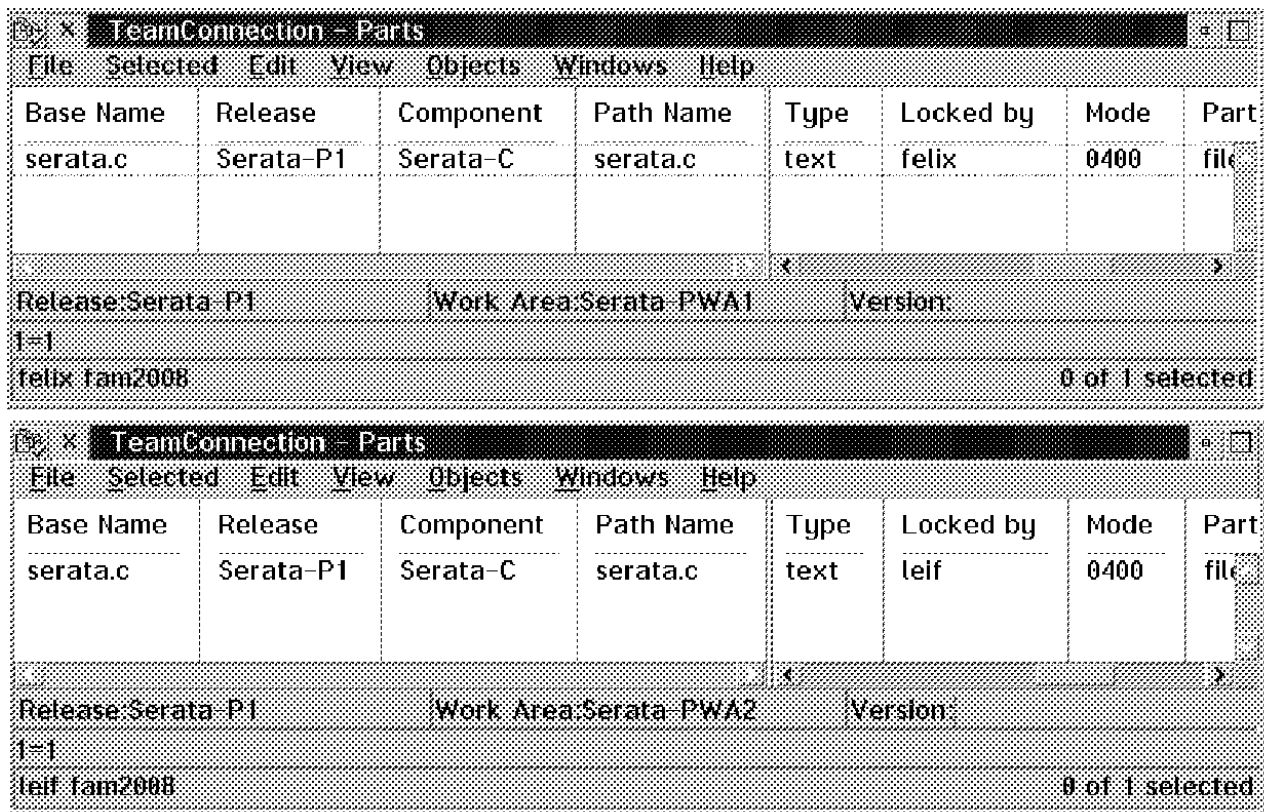


Figure 164. Work Area Serata-PWA1 and Work Area Serata-PWA2

After having completed the changes to the file, user *felix* checks in the file again. As this is the last change to the *Serata-PWA1* work area, *felix* decides to *integrate* the work area.

When user *leif* is done with his changes, he also checks in the *serata.c* file again. He also decides to *integrate* his work area, but in trying to do so, he gets the message shown in Figure 165; that is, he has to refresh the *Serata-PWA2* work area.

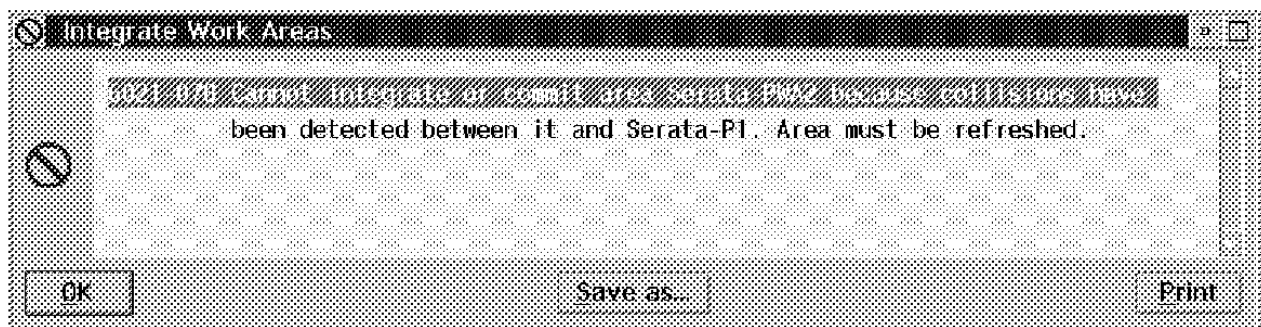


Figure 165. Integrate Work Areas Information Window

Use the **Refresh Work Areas** window (see Figure 166) to refresh the work area. The refresh updates the parts in the work area with any changes from the release, or it refreshes the parts in the work area with any changes from another work area. The refresh also freezes the work area if it is not already frozen.

If you specify a source work area, TeamConnection updates the parts in your work area as follows:

- ☐ For parts that have not been modified in the source work area and have not been modified in the work area you are refreshing, TeamConnection

- does not change the parts.
- For parts that have not been modified in the source work area and have been modified in the work area you are refreshing, TeamConnection does not change the parts.
- For parts that have been modified in the source work area and have not been modified in the work area you are refreshing, TeamConnection links the file in the source work area into the work area you are refreshing.
- For parts that have been modified in the source work area and have been modified in the work area you are refreshing, TeamConnection generates a collision record.

If you do not specify a source work area, TeamConnection updates the parts in the work area you are refreshing with any changes from the release. None of the modified objects in the work area and none of the objects built as a result of your modifications is overwritten by the refresh.

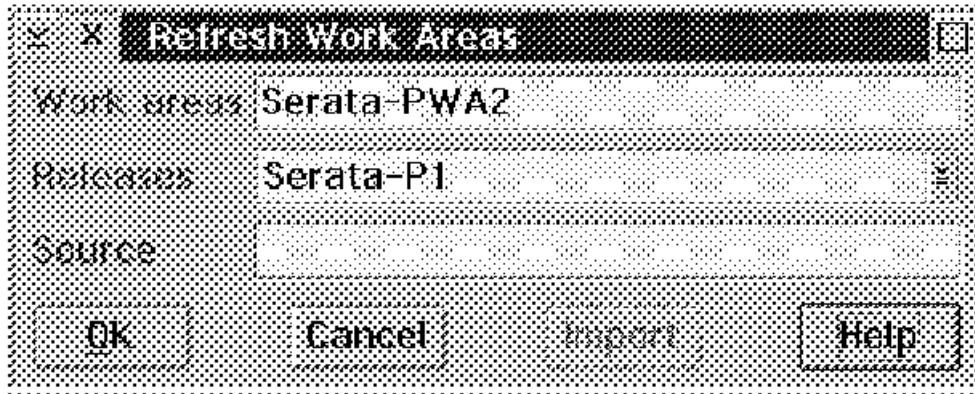


Figure 166. Refresh Work Areas Window

Command Line Interface Syntax

The following is the command line interface syntax for refreshing a work area:

```
teamc workarea -refresh Name ... -release Name ...
               -family Name [-source Name]
               [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-refresh <i>Name ...</i>	Refreshes the contents of one work area with another work area. This command also freezes the work area.
-release <i>Name ...</i>	The releases for which this command is being issued (environment variable: TC_RELEASE)
-family <i>Name</i>	Specifies the family for which this feature is being opened (environment variable: TC_FAMILY)
-source <i>Name</i>	Specifies the name of the source work area during the refresh

-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message after you issue this command

In our case, the command syntax would look like this:

```
teamc WorkArea -refresh Serata-PWA2 -release Serata-P1
```

If you are working in concurrent development mode, collisions can occur during the refresh. If a collision occurs, TeamConnection generates a collision record. You can now build your application within your work area to determine whether your changes integrate with the rest of the release. By doing this, you verify that your changes will not cause problems when you integrate them with the release.

When refreshing the *Serata-PWA2* work area from the *Serata-P1* release, user *leif* gets the message shown in Figure 167. This message indicates that all parts from the release were not refreshed, so user *leif* has to *reconcile* the *collision records* in order to refresh all parts.

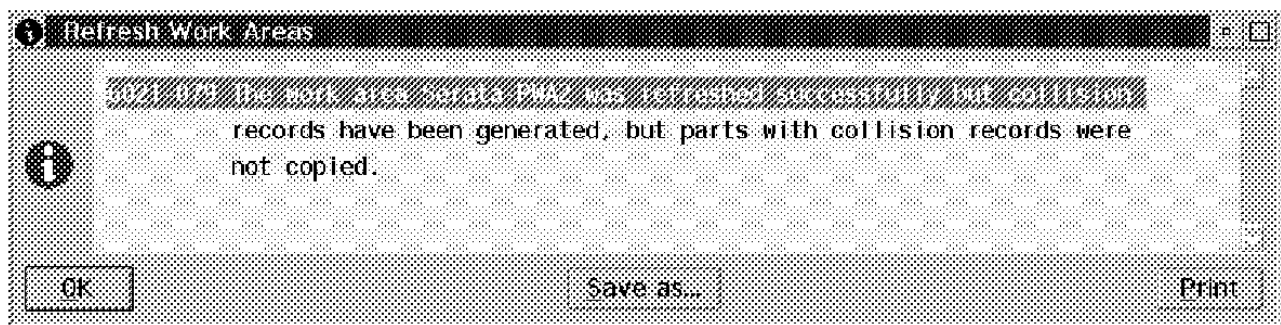


Figure 167. Refresh Work Areas Information Window

To reconcile the *collision records*, user *leif* selects **Show > Collision records > Active** from the pop-up menu in the **TeamConnection - Work Areas** window. These selections bring up the **TeamConnection - Collision Records** window (see Figure 168). You use collision records to decide whether to do one of the following:

- ☐ Accept the parts that already have been committed by another user (thus rejecting your changes)
- ☐ Reconcile or merge aspects of your part changes with the part changes in the committed parts. You can use the TeamConnection merge tool to do this.
- ☐ Reject the committed version of the part and replace it with your part.

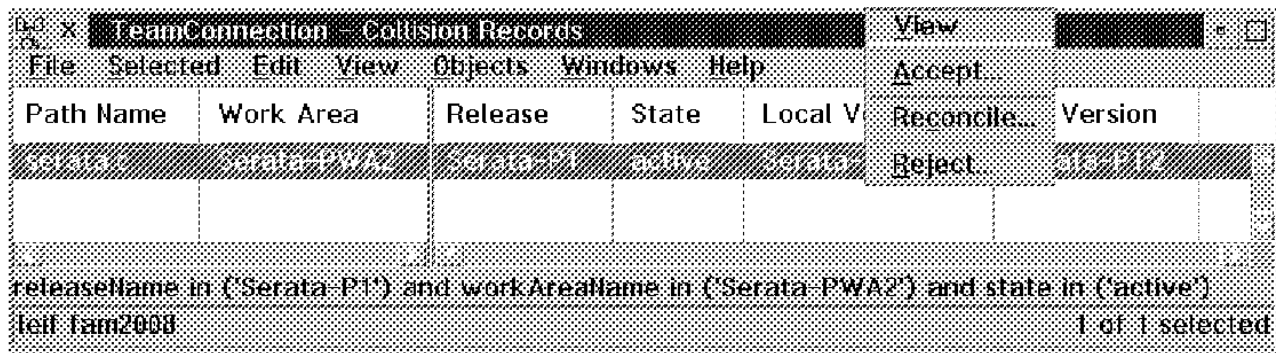


Figure 168. TeamConnection - Collision Records Window

Use the **Reconcile Collision Record** window (see Figure 169) to reconcile the changes you made to a part in your work area with the changes in the committed version of the part. After you enter the required fields in this window, TeamConnection invokes the merge tool. You can then decide which sections of your part and which sections of the committed part are merged into a new part.

The following fields and push buttons are available in the **Reconcile Collision Record** window:

Path name Type the names of the parts under TeamConnection control, including the path name.

Part Types Type the part type. For example, *File* is a part type. Parts with a part type of *File* can exist on the workstation. If you do not specify a part type, this field defaults to *File*. Depending on your development environment, you may have parts that have a part type other than *File*.

Release Type the name of the release with which the parts are associated.

Work area Type the name of the work area with which the parts are associated.

Alternate version Type the version of the part that another user has committed to TeamConnection.

Merge with alternate version Use the following field to specify the merge tool you want to use to reconcile the collision records:

Merge command Type the command to start the merge tool you want to use.

Merge Select **Merge** to start the merge tool to reconcile collision records.

OK Processes the information that you typed and closes the window.

Cancel Does not process the information that you typed and closes the window.

Import Imports text into the field that has cursor focus.

Help Displays help information about the window.

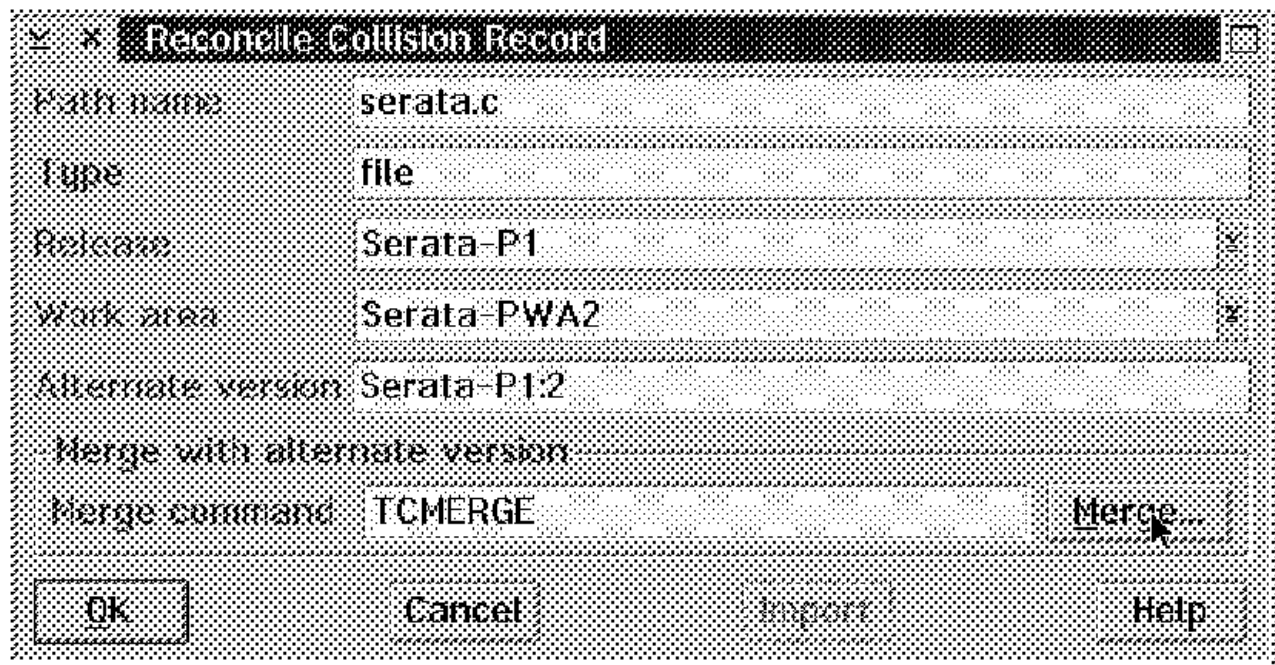


Figure 169. Reconcile Collision Record Window

Command Line Interface Syntax

The following is the command line interface syntax for reconciling collision records:

```
teamc collision -reconcile -path Name -altversion Name -release Name
               -workArea Name ... -family Name [-top Name] [-type Name]
               [-become Name] [-verbose]
```

Attribute Flag and Argument	Description
-reconcile	<p>The latest part that is not integrated takes precedence over the committed part, only if a new version of the part has been checked in since the TeamConnection product detected the collision.</p> <p>When you reconcile parts by using the TeamConnection command line, you must go through the following steps:</p> <ol style="list-style-type: none"> 1. Check out the part in the area that has the collision. 2. Extract the part specified for the alternate version to a different directory from the checked out part.

	<p>3. Run the merge program against the two parts.</p> <p>4. Check in the resultant file.</p> <p>On the GUI, the reconcile command automatically does the preceding steps for you.</p>
-path <i>Name</i>	The path of the part associated with the collision record
-altversion <i>Name</i>	An ID of the view of the system containing the conflicting version of the part
-release <i>Name</i>	The release for which this command is being issued (environment variable: TC_RELEASE)
-workarea <i>Name</i>	The work area associated with the collision record (environment variable: TC_WORKAREA)
-family <i>Name</i>	Specifies the family for which this feature is being opened (environment variable: TC_FAMILY)
-top <i>Name</i>	Specifies the leading portion of the path name that is a subset of the current working directory on the client machine (environment variable: TC_TOP)
Attribute Flag and Argument	Description
-type <i>Name</i>	The type of the parts. The default is <i>File</i> .
-become <i>Name</i>	Identifies the user ID from which you want to issue TeamConnection commands, if the user ID differs from your login. You assume the access authority of the user ID you specify (environment variable: TC_BECOME).
-verbose	Specifies that you want to receive a confirmation message

```
| after you issue this |
| command              |
+-----+
|
```

In our case, the command syntax would look like this:

```
teamc Collision -reconcile -path serata.c -type file
               -release Serata-P1 -workarea Serata-PWA2
               -altversion Serata-P1:2
```

We decide to *merge* the two copies of the *serata.c* file for which the collision record has been created. We do this by clicking on the **Merge...** button. When we click on the **Merge...** button, TeamConnection executes the following comands:

```
teamc Part -stdout -checkout serata.c -release Serata-P1 -workarea Serata-PWA2

teamc Part -stdout -extract serata.c -release Serata-P1 -version Serata-P1:2

TCMERGE E:\VACPP\TMP\2A5VTGEU.C2T E:\VACPP\TMP\JVEAH4RJ.C2T
-out E:\VACPP\TMP\serata.c -titleA Serata-PWA2 -titleB Serata-P1:2 -nologo
```

So TeamConnection:

- ☐ Checks out *serata.c* from work area *Serata-PWA2* (which we refreshed) to standard out
- ☐ Extracts the integrated version of *serata.c* from release *Serata-P1* (the version that has already been integrated) to standard out
- ☐ Invokes the merge tool by calling the *TCMERGE* utility

By looking at the parameter string for the invocation of *TCMERGE* we see that *TCMERGE* actually creates a unique name for both the checked out version and extracted version of *serata.c*:

2A5VTGEU.C2T

Represents the checked out version of *serata.c* from work area *Serata-PWA2*

JVEAH4RJ.C2T

Represents the extracted version from release *Serata-P1*

We also notice that the composite file is called *serata.c*, and all files are located in the *\VACPP\TMP* directory on the *E* drive.

The merge tool can of course be used to compare any text files, not only those that are stored in TeamConnection.

Command Line Interface Syntax

The following is the command line interface syntax for invoking the merge tool:

```
TCMerge [FileA] [FileB] [-out OutFile] [-replace]
        [-titleA Title] [-titleB Title] [-nologo]
```

Attribute Flag and Argument	Description
-FileA	The first file you want to compare. If you do not supply it, you are prompted for it.
-FileB	The second file you want to compare. If

	you do not supply it, you are prompted for it.
-out <i>OutFile</i>	An output file for the composite. If you do not supply it, you are prompted for it when you first attempt to save the composite.
-replace	If the output file exists, replace it.
-titleA <i>Title</i>	A title to use for the first file. If you do not supply it, the full file name is used.
-titleB <i>Title</i>	A title to use for the second file. If you do not supply it, the full file name is used.
-nologo	Do not display the TeamConnection logo.

10.1.4 Using the Merge Tool

TeamConnection Merge is a program for comparing and merging files. It compares two files on disk and notes the lines that are common to both as well as the line differences between the files.

If you do not specify the source files when you start TeamConnection Merge, the first window you see will be the **Open Files** window. Use this window to specify the source files you want to merge or compare.

When TeamConnection Merge compares the source files, it uses color to identify both the matching and the unique blocks of text it finds.

When the main window for TeamConnection Merge first appears (see Figure 170), it displays the results of the merging of the source files. This is known as the composite file. You can select to view either of the two files you are comparing or the composite file that TeamConnection Merge creates. The default view is of the composite file. Standard clipboard functions can be used for simple editing of the composite file.

To the right of the file, TeamConnection Merge displays a bar chart, giving you an overall view of both the matching and unique lines of text in the two files. This is a graphical representation of the comparison of the two files. The bar chart does not identify changed, inserted, or excluded lines in the composite file.

A line immediately below the file is the information area. The information area describes whether the current line is unique to File A, unique to File B, or common to both files. The current line is the line at the cursor location. You can select to have TeamConnection Merge either display the information area or not. The default is to display the information area.

Note: TeamConnection Merge recognizes UNIX-style files (files using only a newline character to mark the end of a line, rather than OS/2's carriage return - newline combination).

The following menu bar items are available from the **TeamConnection Merge** main window (see Figure 170):

File

The File menu contains the following items:

- Open** Opens new source files
- Save composite** Saves the composite file. If you have not previously saved the composite, the **Save Composite As** window appears.
- Save composite as...** Saves the composite file to a new file
- Rebuild composite** Rebuilds the files currently in memory. This action overwrites the composite file, so any unsaved changes are lost.
- Exit** Closes TeamConnection Merge.

Edit

Use the Edit menu to work with the composite file. Any changes you make are made only to the composite file. The source file displays do not change. The Edit menu is available only when you are viewing the composite file. You cannot use TeamConnection Merge to change the source files.

If you want to toggle or view the excluded lines, you must have Show excluded lines active. If you do not have Show excluded lines active, you cannot see the lines after they have been excluded. Excluded blocks are marked with an x at the beginning of the lines; included blocks are marked with a vertical bar.

The Edit menu contains the following items:

- Copy** Copies all marked lines to the clipboard
- Paste** Pastes lines from the clipboard after the current line
- Include >** Select **Include >** to include blocks, lines, File A, or File B.

Note: If you do not have Show excluded lines

selected, this option is not available.

Block	Selects the current block to be included in the composite file
Line	Selects the current line to be included in the composite file
File A	Includes all lines that are unique to file A
File B	Includes all lines that are unique to file B

Exclude >

Select **Exclude >** to include blocks, lines, File A, or File B.

Note: If you do not have Show excluded lines selected, this option is not available.

Block	Selects the current block to be excluded from the composite file
Line	Selects the current line to be excluded from the composite file
File A	Excludes all lines that are unique to file A
File B	Excludes all lines that are unique to file B

Toggle >

Select **Toggle >** to toggle the current block or line.

Note: If you do not have Show excluded lines selected, this option is not available.

Block Toggle the block in the composite file. If the block is excluded, include it. If the block is included, exclude it.

Note: This can also be accomplished by placing the mouse pointer over the block and clicking mouse button 2.

Line Toggles the current line in the composite file. If the current line is excluded, include it. If the current line is included, exclude it.

Note: This can also be accomplished by placing the mouse pointer over the line, pressing Shift, and clicking mouse button 2.

Mark >

Select **Mark >** to mark a line, a block, or all lines.

Block	Marks the current block in the composite
Line	Marks the current line in the composite
All	Marks all lines in the composite

Unmark >

Select **Unmark >** to unmark a line, a block, or all lines.

Block	Unmarks the current block in the composite
Line	Unmarks the current line in the composite
All	Unmarks all lines in the composite

Add lines

Adds lines to the composite at the current location

Edit line

Edits the current line in the composite

View

The View menu contains the following items:

File A

Displays source file A

File B

Displays source file B

Composite

Displays the composite file

Next block

Moves the cursor to the beginning of the next block

Previous block

Moves the cursor to the beginning of the previous block

Start of block

Moves the cursor to the start of the current block

End of block

Moves the cursor to the end of the current block

Options

Use the Options menu to set the comparison options and display characteristics of TeamConnection Merge. Active options are identified in the pull-down list by a check mark. To change the status of an option, just select it.

The Options menu contains the following selections:

Ignore leading blanks

Ignores white space at the beginning of lines. As a result, any indenting of programs becomes irrelevant.

Ignore trailing blanks

Ignores white space at the end of lines

Ignore all blanks

Disregards all white space, wherever it is in the files

Note: Different editors handle white space (for example, spaces, tabs, or nulls) differently. By ignoring this white space, you can achieve more consistent results when TeamConnection Merge compares the files. This is useful for comparing programs in free-format languages, especially if different editors have been used to create the files.

Selecting any of the options listed above causes TeamConnection Merge to rebuild the composite file. Unsaved changes to the current composite file are lost. If you have not saved your changes, you get a message asking you to confirm rebuilding the composite and losing your changes.

Case Sensitive

Determines whether TeamConnection Merge distinguishes between uppercase and lowercase. If this option is active, uppercase and lowercase alphabetical characters are treated as different characters, for example, an uppercase A is not treated the same as a lowercase a. Selecting this option causes TeamConnection Merge to rebuild the composite file. Unsaved changes to the current composite file are lost.

Interleave unmatched lines

Orders unmatched blocks in the composite file so that lines from files A and B alternate. If Interleave unmatched lines is not active, unmatched blocks are displayed as whole blocks from file A or B. Selecting this option causes TeamConnection Merge to rebuild the composite file. Unsaved changes to the current composite file are lost.

Show line numbers

Shows the line number of each line. In the composite file, the numbers for matched lines are taken from file A. For lines that have been pasted into the composite, a line number of 0 is used.

Show excluded lines

Shows the lines you have excluded from the composite file. Excluded lines are marked with an x at the beginning of the line; included lines are marked with a vertical bar. This option is valid only when you are working with the composite file. If you want to toggle or view the excluded lines, you must have Show excluded lines active. If you do not have Show excluded lines active, you cannot see the lines after they have been excluded.

Show information area

Select Show information area to display the information area. The information area describes whether the current line is unique to File A, unique to File B, or common to both files.

Set colors

Brings up the Set colors window

Set font

Brings up the Set font window

Set tab equivalent

Brings up the Set tab window

Help

The Help menu contains the following items:

Help index

Displays a list of selectable help topics

General help

Displays a brief overview of TeamConnection Merge

Using help

Explains the use of the help functions

Help contents

Lists the titles of all help topics

Product information

Displays brief information about TeamConnection Merge

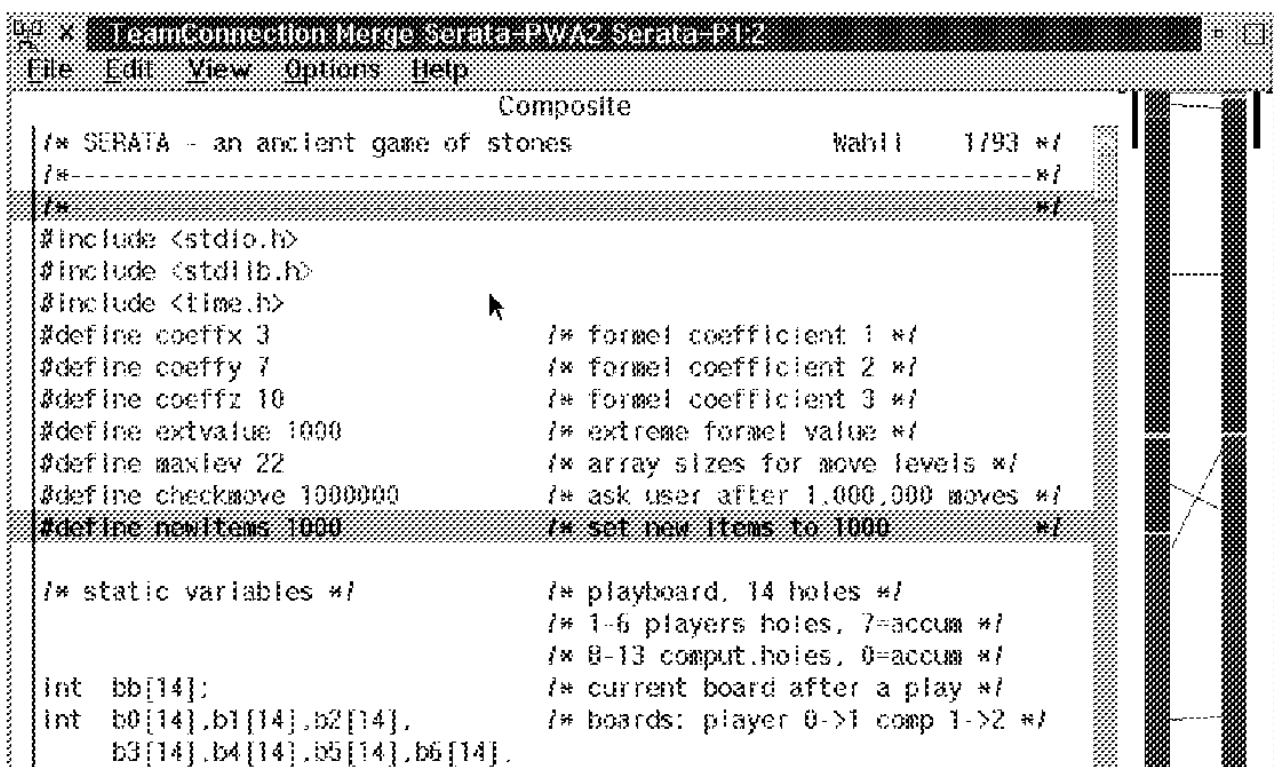
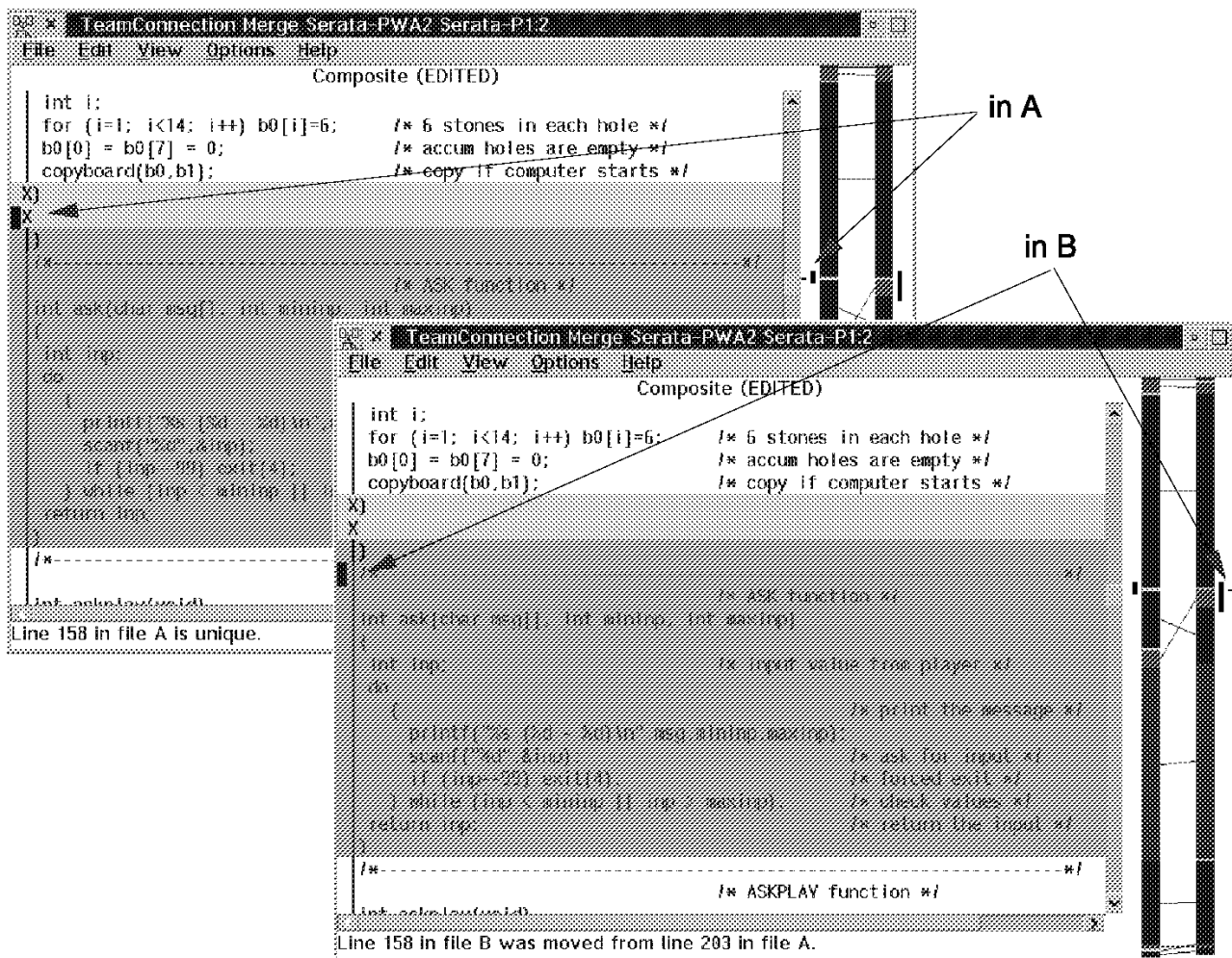


Figure 170. Merge Tool Main Window

In Figure 171 we show how, with the help of the graphical bar at the right of the Merge tool window, you can tell in which file the highlighted area is located. The small dot beside the vertical bar shown in the two first windows in Figure 171 indicates in which file, when using the composite view, the cursor in the main window is positioned, that is, whether it is in file A, B, or both (which is shown in the third window in Figure 171). You can also easily see if portions of the files have changed place between version A and B. Matching portions are shown by colored areas in the graphical bars connected by a thin line.



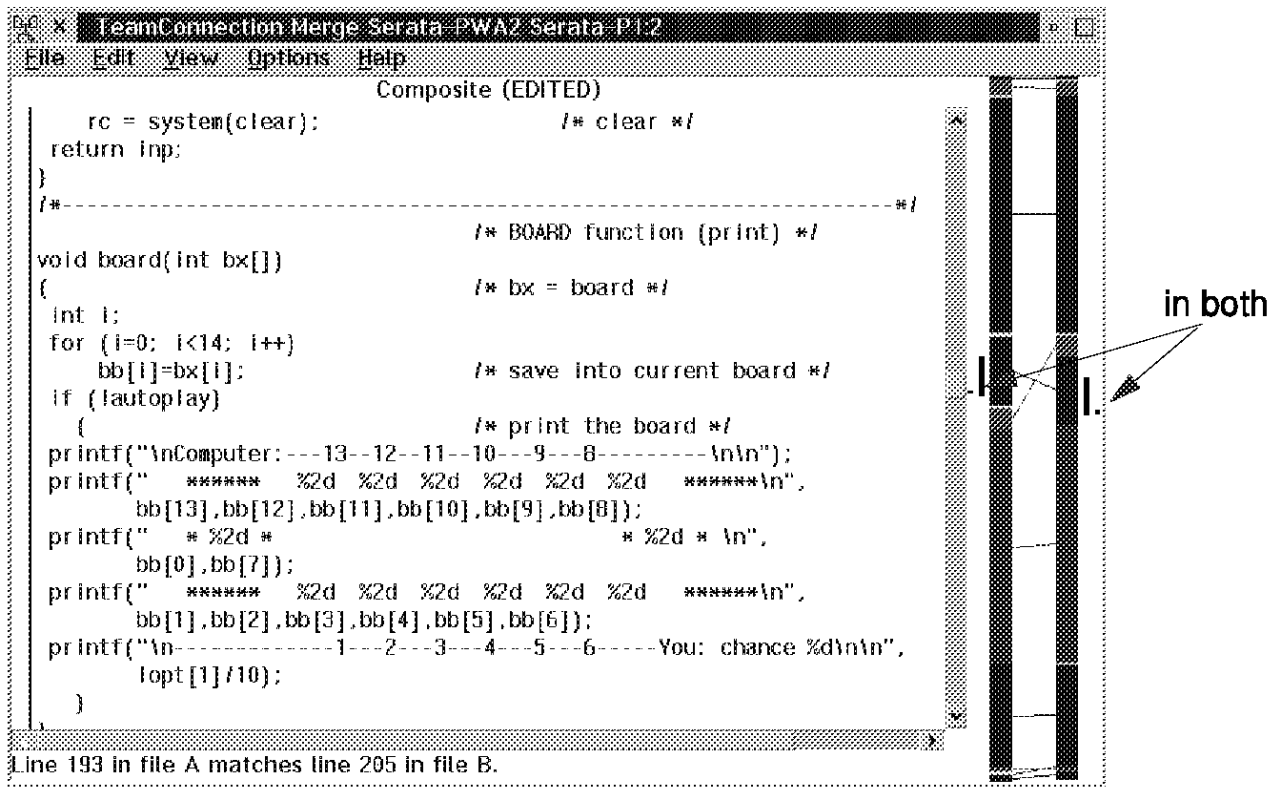


Figure 171. Using the Merge Tool

When we are done with the merge, we save the composite file and exit the Merge tool. When we exit the Merge tool, we are presented with the **Finished Merging Part** window (see Figure 172).

Use the **Finished Merging Part** window (see Figure 172) to specify what you want done with the file that you just finished merging. From this window, you can choose to do nothing with your changes or check the part into TeamConnection.

The following fields and buttons are available in the **Finished Merging Part** window:

Overview

Check it in

Select this radio button to indicate that you want to check in the part.

Unlock it

Select this radio button to indicate that you want to unlock the part without checking in your changes.

Path name

Type the path name for the part that you just finished editing. The path name can include both the directory name and base name.

Release

Type the name of the release that is associated with the part.

Work areas

Type the name of the work areas for the part. The information you type determines which work area is associated with the view.

Directory

Type the source directory of the file you just edited.

Note: The default for this field is the TMP directory defined in your CONFIG.SYS file. This is to avoid corrupting any files in the working directory that might be using the same name.

Check in

Specify the following information if you want to check in the part:

Common releases

Type the names of any common releases. If the parts are common to multiple releases and you want them to remain common, specify any other releases for which

you want the parts to be checked in. You can import selected releases from the **Releases** window.

Remarks

Type comments directly into this field.

Edit

You can also select the **Edit...** push button to type lengthy remarks or to insert text from a file.

Retain lock

Select this check box to specify that you want to perform all of the check in actions but do not want to unlock the parts and allow other users to check them out.

Break common link

Select this check box to break the common link in all releases in which the parts are common except for those specified in the Release and Common releases fields. If you break the common link, the part becomes a shared part among the releases in which the link was broken. Breaking the common link requires authority beyond *PartCheckIn* authority; you must be the component owner or have *PartForceIn* authority, which is defined in the component associated with the part.

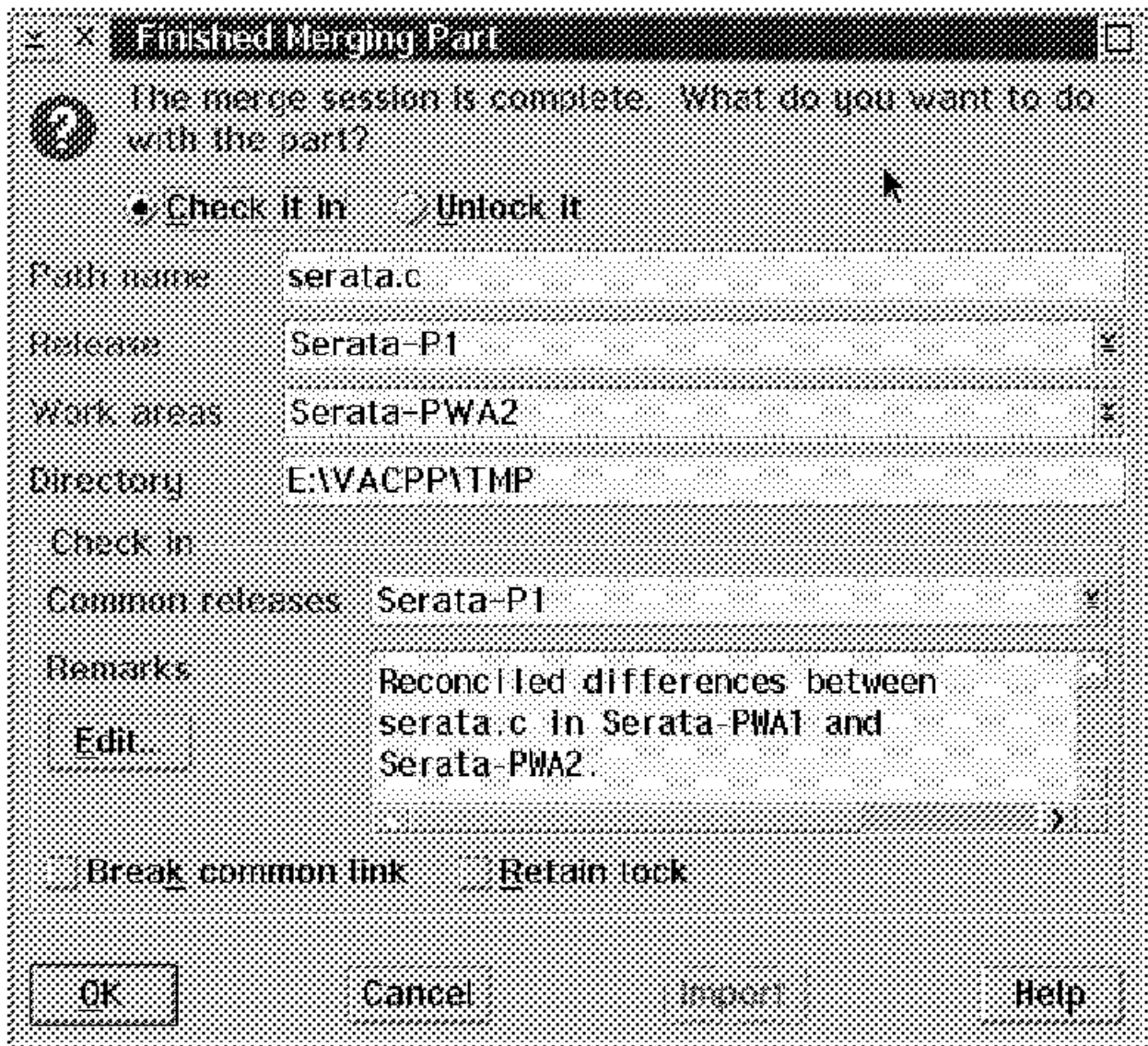


Figure 172. Finished Merging Part Window

When the merged *serata.c* has been checked in again, we can go back to the **Reconcile Collision Records** window (see Figure 169 in topic 10.1.3) and complete the reconciliation by clicking on **OK**. After we have reconciled the collision records, they will disappear from the

TeamConnection - Collision Records window (see Figure 173).

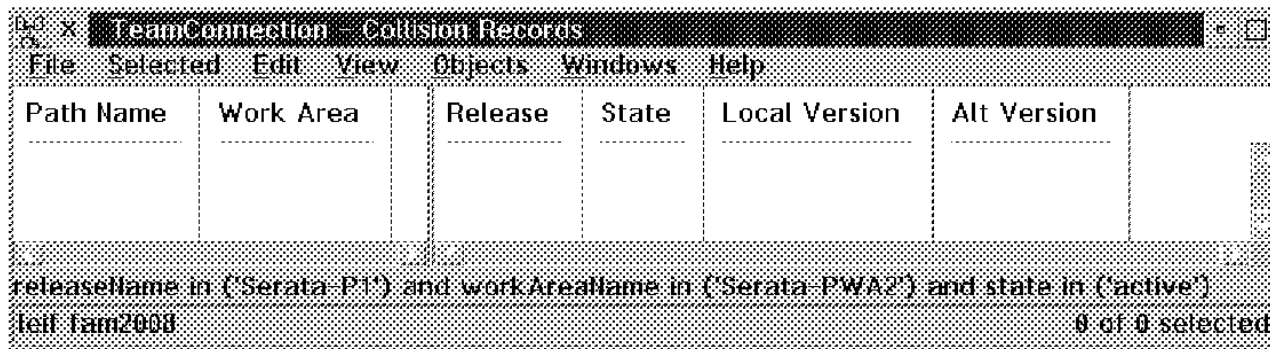


Figure 173. TeamConnection - Collision Records Window after Reconciliation

When all *collision records* have been reconciled, we can continue with the integration of the *Serata-PWA2* work area. After the *Serata-PWA2* work area has been integrated with the release, the state of the work area changes from **fix** to **complete** (see Figure 174).

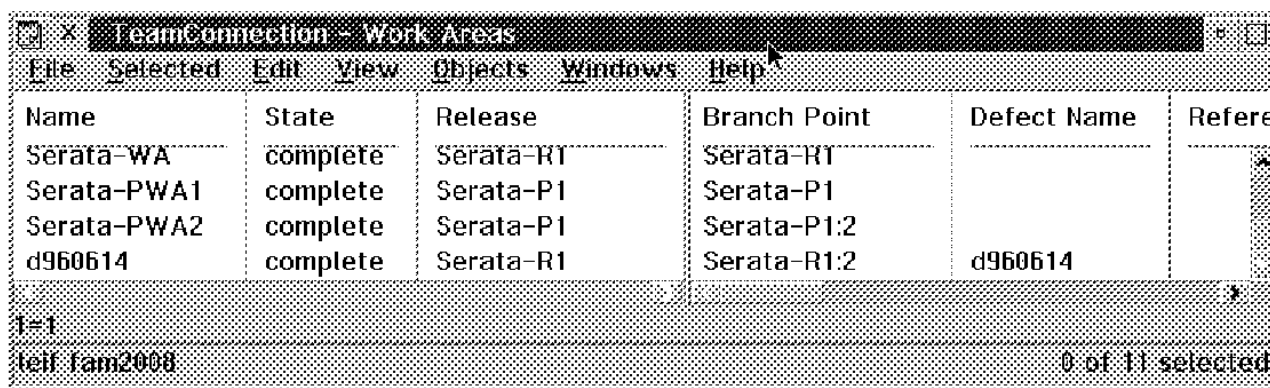
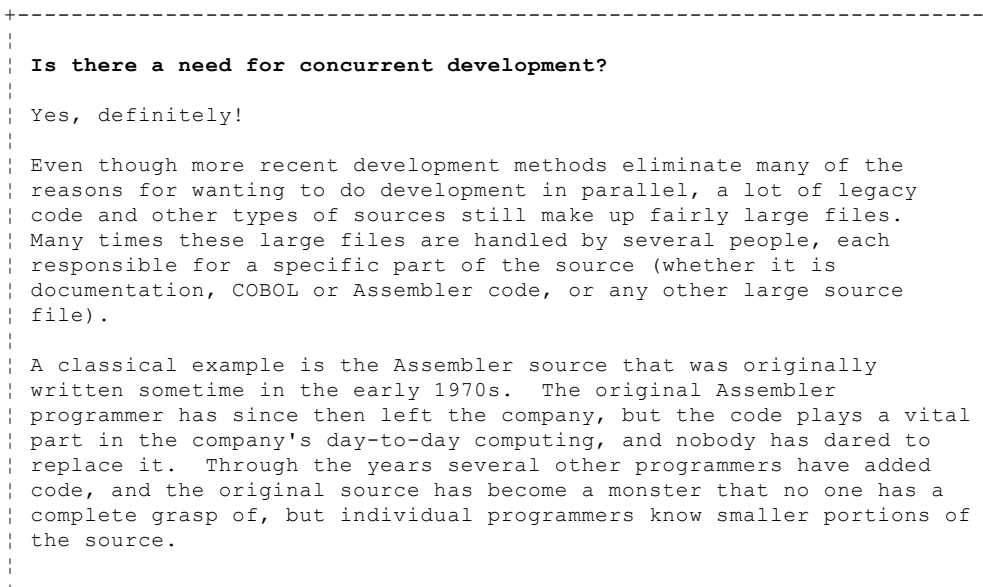


Figure 174. TeamConnection - Work Areas Window after Integration of Serata-PWA2



10.2 Using Configurable Fields

The family administrator can configure fields so TeamConnection more closely matches the development environment. Your development group can store information that is customized to your environment and terminology by doing the following:

- ☐ Changing existing configurable fields
- ☐ Creating new fields

For example, you might want to add a field called *PubImpact* to the defect and feature tables. Programmers can then use this field to notify the writing team as to whether or not a defect or feature affects the accuracy of the product documentation.

Before you configure new fields, you must decide which values are acceptable entries for the fields. To do this, specify a field type when you create the field. If you do not want to use an existing field type, you must create the type before you create the field.

For example, if you create a *PubImpact* field, you might want to create a new field type called *PubImpact* (it can have the same name as your field). If you assign the attributes of *yes*, *no*, and *maybe* to this field, writers can access the user interface or issue a command to get a list of all defects or features that affect the publications. If you also add an attribute of *done*, the writers can indicate when they have finished updating the documentation.

Subtopics

10.2.1 Defining Configurable Field Types

10.2.2 Changing or Creating Configurable Fields

10.2.3 Displaying Configurable Field Properties

10.2.1 Defining Configurable Field Types

For the defect and feature tables, IBM ships configurable field types with defined values. You can use the default field types or change their attributes. For example, IBM defines a field named *Severity*. This field has valid values of 1, 2, 3, and 4. You could add an additional value of 5, or you could change the description of what value 2 represents.

You can also create new configurable field types for the defect, feature, part, or user tables. This allows you to structure problem tracking information for your development environment.

You can create new types or change the attributes of the existing types at any time during your development cycle.

IBM recommends that you use the Family Administrator GUI to create or modify configuration types.

10.2.2 Changing or Creating Configurable Fields

Default configurable fields are shipped by IBM and are installed when the TeamConnection server is configured. If you do not want to use these defaults, you can change them at any time after the family database is created.

The following conditions apply to the use of configurable fields:

- ☐ Each user and part object can have up to 20 configurable fields. Defect and feature objects can have a combined total of 20 configurable fields.
- ☐ Configurable fields cannot be deleted, but they can be renamed.
- ☐ Fields for defect, feature, and user objects are effective only on the create, open, and modify actions.
- ☐ Fields for the part object are effective only on the modify action.
- ☐ Defined values for a configurable field cannot exceed 85 characters and cannot contain spaces.
- ☐ Only character data can be entered in configurable fields.
- ☐ You can use the data from configurable fields to search the database and display information in reports, but TeamConnection does not use the data. For example, if you have a field called *PubImpact*, TeamConnection cannot change the state of a defect based on the value of this field, but users can sort all defects and features by whether or not they impact the publications.
- ☐ When you add fields, TeamConnection displays them on the GUI like any predefined field. However, the help information for configuring fields for the GUI and the commands do not reflect your new or changed fields.

10.2.3 Displaying Configurable Field Properties

To display the properties of the active configurable fields for an object, type one of the following commands from an OS/2 prompt:

- **teamc defect -configInfo -family** *familyName* [-raw]
- **teamc feature -configInfo -family** *familyName* [-raw]
- **teamc part -configInfo -family** *familyName* [-raw]
- **teamc user -configInfo -family** *familyName* [-raw]

These commands show you exactly what has been defined and let you verify that the fields were loaded correctly.

If the -raw flag is used, the information is organized in a fixed ASCII table format as follows:

Field Label]Title]Attribute]DB Column Name]Create]Required]Field Type

Note: The properties of both the *Prefix* and *Severity* nonconfigurable fields are displayed for defects, whereas only the *Prefix* field is displayed for features.

10.3 Using Report Facilities

TeamConnection users can view or print reports about an object. When you create a field, TeamConnection adds the new field to the report. You can choose the field information to present to the user and the place on the report where the information appears.

Reports are displayed in two formats: stanza and table.

Use the system editor to edit the files listed below and manually change the position of report fields on the reports TeamConnection generates for the user, defect, feature, and part objects. You can also change or delete the format specification. Before you change the report formats, make backup copies of the *.fmt* files.

- ☐ *cfgfield\defect.fmt*
- ☐ *cfgfield\feature.fmt*
- ☐ *cfgfield\part.fmt*
- ☐ *cfgfield\user.fmt*

Each *.fmt* file is divided into the following five sections, separated by colons:

- ☐ StanzaViewFormat
- ☐ StanzaViewColumn
- ☐ TableViewFormat
- ☐ TableViewColumn
- ☐ TableViewHeader

The column sections describe the column name of each of the labels specified in the format sections. The header section specifies how the columns appear in the table format.

The format sections specify the layout of the report. For example, a format specification of **%3\$-25.25s** indicates the following:

- %** The start of the format specification
 - 3** The sequence number of the field that is generated by TeamConnection. The dollar sign must appear after the sequence number.
 - The output is left-justified. If you do not include this character, the output is right-justified.
 - 25** The minimum number of characters (bytes) of output
 - .25** The maximum number of characters (bytes) printed for all or part of the output field, or minimum number of digits printed for integer values
- If you do not want the field displayed, type **0.0**. For example, assume you have three sequence fields: 1, 2, and 3. If you do not want sequence 2 displayed, type:
- %1\$-4.4s %2\$-0.0s %3\$-15.15s**
- s** Type of data:
 - s** for strings
 - ld** for integers

You can specify only a data type of **s** for configurable fields. Use **ld** to display existing values, such as defect age.

10.4 User Exits

User exits are not necessary for the operation of TeamConnection; they are optional and can be configured for each family.

With user exits, you can specify additional actions to be performed before completing or proceeding with a specific TeamConnection command action. A user exit enables the TeamConnection server to call a user-defined program during the processing of TeamConnection actions. The program can be an executable file or a command file. Thus, you can use TeamConnection as a trigger to start non-TeamConnection processing. You can also use user exits to restrict certain TeamConnection actions based on external considerations. For example, you might have a user exit scan C source files to ensure that the source code conforms to the standards defined by your development process.

The userExit file indicates the programs you want started for specific TeamConnection actions. You can add entries to this file by using the Family Administrator GUI (the recommended method) or by directly editing the file.

User exits are provided for most TeamConnection actions. The actions that support user exits are listed in Appendix D, "User Exit Parameters," of the *IBM TeamConnection for OS/2 User's Guide*.

Subtopics

10.4.1 Writing User Exit Programs

10.4.2 Suggestions

10.4.1 Writing User Exit Programs

Follow these guidelines to write user exit programs:

- ☐ Limit the length of time that the user exit program runs.
- ☐ Avoid using TeamConnection commands that update the database. Instead, use the Report command or commands that have the -view action.

Note: Using commands that update the database can cause deadlock.
- ☐ Run at least two server daemons when using TeamConnection commands from within a user exit.
- ☐ User exit programs do not permit user interaction (for example, from a user exit program, you cannot prompt a user with a read command).
- ☐ Define only one user exit program for each TeamConnection action and exit ID combination. If you define more than one program, TeamConnection uses the last one you define.

When you want TeamConnection to start a user exit, you must associate the user exit program with TeamConnection actions. IBM recommends that you use the Family Administrator GUI to associate the actions with the program.

10.4.2 Suggestions

This section suggests some user exit programs for the TeamConnection actions used most frequently. Many other user exits may be required by your software development process, company policies, or project practices.

Some ideas associated with the problem tracking process include:

- ☐ Start up a *quality* inspection process when the state of a defect or feature moves to the **review** state.
- ☐ Create a clone defect or feature if a test or verification record is rejected.
- ☐ Supply quality database and tools with defect and feature attributes to calculate some metrics.

The following are some suggestions for user exit programs linked to both TeamConnection *PartCheckIn* and *PartCreate* actions:

- ☐ Run some tools to analyze whether the programming rules have been respected. For example, ensure that no "goto" instructions are used or check the number of nested *if* statements.
- ☐ Compute the number of lines of code that have been developed.

```
+--- CAUTION -----+
|
| Be careful when using user exits! Too many user exit programs slow
| down TeamConnection response time!
|
| User interactions are not allowed in a user exit.
|
+-----+
```

Chapter 11. Maintaining Your TeamConnection Environment

11.0 Chapter 11. Maintaining Your TeamConnection Environment

In this chapter we explain how to:

- ☐ Change the age of defects and features
- ☐ Resolve TeamConnection errors
- ☐ Maintain the TeamConnection database

Subtopics

11.1 Changing the Age of Defects and Features

11.2 Resolving TeamConnection Errors

11.3 Maintaining the TeamConnection Database

11.1 Changing the Age of Defects and Features

TeamConnection provides two aging utilities: **Age** and **resetAge**. Use these utilities to update the age value of defects and features while work is in progress. If you do not use these utilities, the age value for each defect and feature remains at zero.

Subtopics

11.1.1 The Age Utility

11.1.2 The resetAge Utility

11.1.1.1 The Age Utility

Use the age utility to increment the age value by 1 for each defect or feature that is in a specified state. The *age.cmd* file is located in the directory where the TeamConnection server is installed. Initially, the file is set up to update the age of defects that are in one of the following states:

- ☐ Open
- ☐ Working
- ☐ Design
- ☐ Size
- ☐ Review

You can edit the file to delete one or more of these states or to add any of the following states:

- ☐ Canceled
- ☐ Returned
- ☐ Closed
- ☐ Verify

Run the age utility from a server machine, using the following command:

```
age familyName
```

where **familyName** is the name of the TeamConnection family.

11.1.2 The resetAge Utility

Use the resetAge utility to reset the age of defects and features based on their state (**open**, **working**, **design**, **size**, and **review**), the date on which they were opened, and the selected aging increment. Run the resetAge utility from a server machine, using the following command:

```
resetage ageIncr
```

where **ageIncr** is one of the following:

fullweek Ages the defects and features according to a 7-day schedule

workweek Ages the defects and features according to a 5-day work week schedule.

Before you run the utility, ensure that the TC_FAMILY environment variable is set to the name of the appropriate family.

11.2 Resolving TeamConnection Errors

The TeamConnection library includes the *Messages* book which lists messages that include recovery information. Use this book to help you resolve TeamConnection messages. If you require further help, refer to TeamConnection's error log or audit log. In this section we explain how to use these error and audit logs and the trace facility.

Subtopics

- 11.2.1 Using the Error Log
- 11.2.2 Using the Audit Log
- 11.2.3 Using the Trace Facility

11.2.1 Using the Error Log

Severe errors that are encountered by the family server are recorded in the *syslog.log* file. Use this file to help you better understand and resolve the error. The *syslog.log* file usually provides more information than what is found in the initial message. There is only one *syslog.log* file, so if you have multiple families, error information for each family is recorded in the same file.

11.2.2 Using the Audit Log

For each family, TeamConnection provides an audit log that contains an entry for every action performed since the family was created. The *audit.log* file is located in the directory where your family database is installed.

Because the *audit.log* file contains information about both successful and unsuccessful transactions, it is useful for determining the source of a problem. It also includes an entry whenever an unauthorized attempt is made to access the TeamConnection server. This can help you audit your system's security.

The following information is recorded in the audit log for each transaction:

- For authorized transactions
 - Process ID number of the family server
 - TeamConnection action
 - Whether the transaction was successful or not
 - Date and time of the transaction
 - User ID of the person who requested the action
 - The name of the host system from which the user is accessing TeamConnection
 - Additional information for successful transactions, or error messages for unsuccessful transactions
- For unauthorized transactions
 - Process ID number of the family server
 - User ID of the person who requested the action
 - The name of the host system from which the user is accessing TeamConnection
 - Notification that the request was unauthorized
 - Date and time of the transaction request
 - Error message

Figure 175 shows an example of information as it appears in the *audit.log* file:

```
+-----+
|
| 02353,Report,SUCCESS,05/31/96,17:41:31,leif,leif,calluna.almaden.ibm.|om,PartView,1=1
| 02353,Report,SUCCESS,06/03/96,17:31:50,leif,leif,calluna.almaden.ibm.|om,WorkareaView,1=1
| 02353,Report,SUCCESS,06/03/96,17:32:24,leif,leif,calluna.almaden.ibm.|om,PartView,1=1
| 02353,Report,SUCCESS,06/03/96,17:33:34,leif,leif,calluna.almaden.ibm.|om,PartView,1=1
| 00076,Report,SUCCESS,06/11/96,15:39:29,leif,leif,calluna.almaden.ibm.|om,WorkareaView,1=1
| 00075,Report,SUCCESS,06/11/96,15:39:47,leif,leif,calluna.almaden.ibm.|om,PartView,1=1
| 00076,PartAdd,SUCCESS,06/11/96,15:40:53,leif,leif,calluna.almaden.ibm|com,2008FULL.PRS,SG26-2008_R1,w20
| 00077,Report,SUCCESS,06/11/96,15:41:00,leif,leif,calluna.almaden.ibm.|om,PartView,1=1
| 00075,PartCheckOut,SUCCESS,06/11/96,15:41:39,leif,leif,calluna.almade|.ibm.com,2008FULL.PRS,SG26-2008_F
| 00077,Report,SUCCESS,06/11/96,16:03:54,leif,leif,calluna.almaden.ibm.|om,Config,1=1 order by name, conf
| 00075,FeatureOpen,SUCCESS,06/11/96,16:17:19,leif,leif,calluna.almaden|.ibm.com,2008FU-1
| 00076,Report,SUCCESS,06/11/96,16:20:20,leif,leif,calluna.almaden.ibm.|om,CompView,1=1
| 00077,CompView,SUCCESS,06/11/96,16:20:40,leif,leif,calluna.almaden.ib|.com,2008_PRS
| 00075,Report,SUCCESS,06/11/96,16:29:41,leif,leif,calluna.almaden.ibm.|om,FeatureView,1=1
| 00076,PartCheckIn,SUCCESS,06/11/96,16:30:35,leif,leif,calluna.almaden|.ibm.com,2008FULL.PRS,SG26-2008_R1
| 00077,PartCheckOut,SUCCESS,06/11/96,16:31:11,leif,leif,calluna.almade|.ibm.com,2008FULL.PRS,SG26-2008_F
| 00075,PartUnlock,SUCCESS,06/11/96,16:32:15,leif,leif,calluna.almaden|.bm.com,2008FULL.PRS,SG26-2008_R1
| 00076,Report,SUCCESS,06/11/96,16:32:44,leif,leif,calluna.almaden.ibm.|om,ReleaseView,1=1
|
+-----+
```

Figure 175. Sample audit.log File

Subtopics

11.2.2.1 Cleaning up the Audit Log

11.2.2.1 Cleaning up the Audit Log

TeamConnection continually appends information to the end of *audit.log*. To keep this file from growing too large, type the following from a command line in the directory where your server is installed:

```
cleanup fileSize
```

where *fileSize* is the size of the specified file in bytes. If you do not specify the size, the default is 256000.

Before issuing this command, stop the family server.

TeamConnection creates a backup file called *audit1.log*. You can rename this file to any name you want for archive purposes. If you do not rename the file, TeamConnection keeps three backup logs in addition to the current log. Each time you run the cleanup program, TeamConnection moves the contents of each log file as follows:

1. *audit3.log* information is moved to *audit4.log*.
2. *audit2.log* information is moved to *audit3.log*.
3. *audit1.log* information is moved to *audit2.log*.
4. *audit.log* information is moved to *audit1.log*.

The *audit.log* file is empty and ready to log new information.

11.2.3 Using the Trace Facility

TeamConnection provides environment variables for trace. Modify the trace environment variables only when directed to do so by an IBM service representative.

Table 8 lists the TeamConnection trace environment variables, their purpose, and the TeamConnection component that uses them.

Table 8. TeamConnection Trace Environment Variables		
Environment Variable	Purpose	Component
BSERV_LOG	Indicates which parts are to be traced for the build agent or processor. You can set this variable when directed to do so by an IBM service representative; otherwise it is set to null.	Build Agent or Processor
BSERV_TRACEATTEMPTS	Specifies maximum number of failed trace attempts accepted before giving up. You should not have to change this variable.	Build Agent or Processor
BSERV_TRACEDELAY	Specifies the amount of time, in seconds, that TeamConnection waits, when a trace attempt for the build agent or processor fails, before attempting another trace. The default is 1 sec. You should not have to change this variable.	Build Agent or Processor
BSERV_TRACEFILE	Specifies the trace file path and name for the build agent or processor	Build Agent or Processor
BSERV_TRACESIZE	Specifies the maximum size of the trace file for the build agent or processor, in bytes. If reached, wrapping occurs.	Build Agent or Processor
TC_TMP	Specifies the directory where the family server stores temporary files	Family Server
TC_TRACE	Specifies the variable that lets the user designate which parts should be traced. Modify this variable only when directed to do so by an IBM service representative. Otherwise it is set to null.	Client Family Server
TC_TRACEATTEMPTS	Specifies the maximum number of failed trace attempts accepted before giving up. Modify this variable only when directed to do so by an IBM service representative.	Client Family Server
TC_TRACEDELAY	Specifies the amount of time in seconds that TeamConnection waits, when a trace attempt fails, before attempting another trace. The default is 1 sec. Modify this variable only when directed to do so by an IBM service representative.	Client Family Server
TC_TRACEFILE	Specifies the output (part path and name) of the trace that the user designates by using TC_TRACE.	Client Family Server
TC_TRACESIZE	Specifies the maximum size of the trace file in bytes. If this size is reached, wrapping occurs. The default is one million bytes.	Client Family Server

11.3 Maintaining the TeamConnection Database

In this section we describe the following ObjectStore utilities, which you can use to maintain and tune your TeamConnection database.

- osbackup** Provides online backup of TeamConnection databases
- oscp** Creates a copy of a TeamConnection database
- osrestor** Restores databases that were backed up by means of osbackup
- ossvrpin** Reports whether the family server is running on a specified host
- ossvrsta** Displays statistics on all clients currently connected to the family server, as well as information about the family server that is not specific to a particular client
- osverify** Verifies that pointers within a database are valid

The *osbackup* and *osrestor* utilities provide protection from catastrophic data loss due to hardware failure. They can also be used to transport databases from one location to another.

Subtopics

- 11.3.1 osbackup
- 11.3.2 oscp
- 11.3.3 osrestor
- 11.3.4 ossvrpin
- 11.3.5 ossvrsta
- 11.3.6 osverify

11.3.1 osbackup

Use the *osbackup* utility to back up your TeamConnection databases. This utility transparently backs up TeamConnection to secondary storage while it is running without affecting concurrency.

If your family uses one or more remote databases in addition to the family database, you must back up all databases simultaneously. Specify the path name of each database on the back up command.

Use the following command to back up your databases:

```
osbackup [options] -f backup_image_file -b [blocking factor] pathname ...
```

where:

- **-f** *backup_image_file* is the output file for the backup image.
- **-b** *blocking factor* is the blocking factor for tape input and output. The blocking factor is in units of 512-byte blocks, and the maximum you can specify is 512 blocks. This parameter is ignored for regular files.
- *pathname* is the path name of the database to be backed up. Wildcards are not supported. You can specify more than one database; however, all databases you specify must reside on the same server. Instead of, or in addition to, this argument, you can use the **-I** option to specify the databases.
- *options*
 - i** *incremental_record_file*
Specifies a file that contains information about which databases have been backed up and when they were backed up. The utility uses this information to determine which segments within a database have been modified since the last backup at a lower level. It then backs up only modified segments. If you do not specify this option, the %os_rootdir%\etc\osbackup.rec default file is used.
 - I** *import_file*
Specifies a file that contains a list of database path names, one name per line, that you want to back up. Leading and trailing blanks are ignored. You can still specify additional path names on the command line.
 - L** *level*
Specifies the level of the backup. Backup is incremental at the segment level, that is, the utility backs up a segment only if it has been modified since the last backup at a lower level. Backup levels of 0 through 9 are supported. A level 0 backup, which is the default, backs up all of the segments in all of the specified databases. Levels 1 through 9 produce incremental backups that save only those segments that have changed since the most recent backup at a lower level.
 - s** *size*
Sets the size, in megabytes, of the volume to which the information is being backed. Use this option when backing up to a tape device, because the end of media cannot be reliably detected on some systems. You are prompted to insert a new tape when the specified size limit is reached.

11.3.2 oscp

Use the *oscp* utility to create a copy of a TeamConnection database. The syntax of *oscp* is:

```
oscp source_pathname destination_pathname
```

The database you specify as the *source_pathname* is copied to the database you specify as the *destination_pathname*. If the destination database already exists, it is deleted before the copy is performed.

Wildcards in the path name are not supported, and you cannot specify a directory.

Before performing the copy, the utility verifies that the database being copied is transaction-consistent and fully up-to-date. When the database is copied, the copy receives a new, unique ID.

11.3.3 osrestor

The *osrestor* utility restores databases that have been backed up with the *osbackup* utility. TeamConnection cannot access databases that are being restored until the entire restore process completes.

If your family uses one or more remote databases in addition to the family database, you must restore all databases simultaneously. Specify the path name of each database on the restore command.

To restore databases, always begin with a full level 0 backup image; *osrestor* prompts for any additional incremental backup images you might want to apply. Not all incremental backups necessarily have to be applied. To determine which incremental backups to apply, list the backup levels in chronological order, starting with the level 0 backup.

For instance, if you made a level 0 backup on Monday, a level 5 on Tuesday and Wednesday, a level 3 on Thursday, and a level 4 on Friday, your list would look like this: 0, 5, 5, 3, 4. Scan the list from right to left and find the lowest incremental backup level greater than 0. In this example, that is the level 3 backup made on Thursday. Therefore, all increments made between the level 0 backup and this level 3 backup need not be applied. To restore databases to their current state as of the backup on Friday, you must apply the level 0 backup and the incremental backups made at level 3 and 4, in that order.

Use the following command to restore your TeamConnection databases:

```
osrestor [options] -f backup_file pathname ...
```

where:

- **-f** *backup_file* is the file or tape device that contains a backup image from which to restore the databases. By default, all databases in the backup image are restored to the current working directory. Use the **-d** option to specify a different directory.
- *pathname* specifies the database you want to restore. The database is restored into the current working directory, or into the directory specified by using the **-d** option. You can specify more than one database name. Wildcards are not supported.

When you do not specify a path name, all databases in the backup image are restored. You must specify the path name if you want the restored databases placed in the root directory.

- *options*

-d *re_directory*

Specifies the directory in which to restore the databases. If you do not specify this option, the current working directory is used.

-n

Restores only the databases contained in the directory specified in the *pathname* argument. If you do not specify this option, all databases in the specified directory and its subdirectories are restored.

-t

Prints a listing of databases contained in this backup image.

11.3.4 *ossvrpin*

Use the *ossvrpin* utility to verify that the family server is running on the specified host. The syntax is:

```
ossvrpin [-v] [hostname]
```

where:

- ☐ **-v** provides more information when the family server cannot be contacted.
- ☐ *hostname* is the name of the host for which you want to test the connection. If not specified, this defaults to the host from which the command is issued.

11.3.5 *ossvrsta*

Use the *ossvrsta* utility to display statistics about all clients that are currently connected to the family server on the specified host. Family server resource information is provided for each client, followed by each client's state, grouped by state. This command also displays the parameter settings and usage meters for the family server. In the report, each client is identified by its host name. If the program name is known, it is listed along with the process ID on that host. If the program name is not known, it is so noted.

The command syntax for *ossvrsta* is:

```
ossvrsta hostname
```


11.3.6 osverify

Use the *osverify* utility to verify that pointers within a database are valid. It verifies that:

- ☐ There are no transient pointers.
- ☐ Persistent pointers point to valid, not deleted, storage.
- ☐ The declared type for a pointer as determined from the schema matches the actual type of the pointed-to object.

The command syntax for *osverify* is:

```
osverify [-o] [-v] [-all] [-w workspace_name]
          [-limit number] pathname
```

where:

- ☐ **-o** prints out the name of every object in the database.
- ☐ **-v** specifies to print every pointer value.
- ☐ **-all** verifies that internal segment 0 is found.
- ☐ **-w workspace_name** verifies the pointers for versioned databases in the specified workspace.
- ☐ **-limit number** limits the number of error messages that are reported for any segment in the database to the number you specify.
- ☐ *pathname* is the path name of the database that you are verifying. Wildcards are not supported.

When *osverify* detects a pointer that is not valid, it indicates the location and the value of the pointer. When possible, it prints out a symbolic path to the bad pointer, starting with the outermost enclosing object.

12.0 Chapter 12. Family Administration

The family administrator can use either the Family Administrator GUI or the command line to configure TeamConnection families. In this chapter we explain how to do the following tasks from the GUI or a command line:

- ☐ Creating or modifying authority groups
- ☐ Creating or modifying interest groups
- ☐ Configuring component or release processes
- ☐ Defining configurable field types
- ☐ Creating or modifying configurable fields
- ☐ Changing report formats
- ☐ Configuring user exits

Doing these tasks from the command line sometimes requires extra steps and is more prone to error. For these reasons, we recommend that you use the GUI.

Subtopics

- 12.1 Creating or Modifying Authority Groups
- 12.2 Creating or Modifying Interest Groups
- 12.3 Configuring Component or Release Processes
- 12.4 Defining Configurable Field Types
- 12.5 Changing Report Formats
- 12.6 Setting Up User Exits

12.1 Creating or Modifying Authority Groups

When a TeamConnection family is created, the authority table is primed with default information contained in the *authorit.ld* file. In this section we provide instructions for manually editing the *authorit.ld* file to create new authority groups or change information about existing authority groups. When you change the *authorit.ld* file, you must also reload the contents of the authority table in the TeamConnection database.

Subtopics

- 12.1.1 Editing the *authorit.ld* File
- 12.1.2 Reloading the Authority Table
- 12.1.3 Using the GUI

12.1.1.1 Editing the *authorit.ld* File

To add new authority groups or to add actions to an existing authority group, edit the *authorit.ld* file from the server machine in the directory where the family's database exists. Add entries to the file, using the following format:

```
AuthorityGroup|ActionName
```

AuthorityGroup

This is the name of an existing authority group or the name of a group that you are creating. The name can be 15 characters long; it cannot contain blanks, tabs, or vertical separators. For an existing authority group, type the name exactly as it appears in the database table. The default names provided by IBM use all lowercase characters.

ActionName

This is the name of an existing TeamConnection action. Specify only one action per entry. You must type the name exactly as it appears in the database table. See the list of actions in Appendix F, "Authority and Notification for TeamConnection Actions" in the *IBM TeamConnection for OS/2 Version 1 User's Guide* for the correct spelling and capitalization. Certain actions cannot be included in an authority group. These actions are noted in the table found in Appendix H, "Worksheets " in the *IBM TeamConnection for OS/2 Version 1 User's Guide*.

12.1.1.2 Reloading the Authority Table

Whenever you change the *authorit.ld* file, you must reload the contents of the authority table before your users can use the new and changed authority groups. You can reload the authority table as often as necessary. We recommend that you stop the family server before you reload the authority table. To reload the authority table, issue the following command from the server machine in the directory where the *authorit.ld* file is stored:

```
fhclauth authorit.ld dbpath\familyname
```

where:

- ☐ *dbpath* is the name of the directory where your family's database exists.
- ☐ *familyname* is the name of your TeamConnection family.

Before issuing this command, ensure that the TC_FAMILY environment variable is set to the correct family name.

To verify that the authority table loaded correctly, use the *report -view* command to generate a report on the authority table. For example, to verify that a new authority group named *general* was added to the table, type the following from a client machine on an OS/2 or TeamConnection command line:

```
teamc report -view authority -where "name='general'"
```

If the table loaded correctly, information about the *general* authority group appears. If the table did not load correctly, make the necessary changes to the *authorit.ld* file and run the *fhclauth* command again. For more information about the *report -view* command, refer to the *IBM TeamConnection for OS/2 Commands Reference*.

12.1.1.3 Using the GUI

Use the **Authorities** page (see "Authority Groups Page" in topic 5.2) to set the authority for each authority group. When you set the authority for each authority group, you identify the actions that the various authority levels are allowed to perform.

When the database is initially created, the authority table contains the default values for the authority groups. If the default authority groups are not adequate for your development organization, you can create new authority groups or modify existing groups. Authority groups can be created or modified at any time during your development cycle.

First, decide which group of actions the intended users are required to perform. See if there is an authority group that closely matches your needs. If there is, you might want to modify the group. Otherwise, you will have to create a new group.

Do the following to create or modify authority groups:

1. Display the family icon's pop-up menu, then select **Settings**. The **Settings** notebook appears.
2. Select the **Authorities** page to display the authority group settings.
3. Do one of the following:
 - ☐ To change an existing group, highlight the group from the **Authority Groups** list, and then select or deselect the appropriate actions from the **Has** authority to list.
 - ☐ To create a new group, type the new name in the entry field, select **New**, and then select or deselect the appropriate actions from the **Has** authority to list. When you type a new name, the actions for the highlighted authority group are highlighted in the **Has** authority to list. Therefore, you might want to highlight an authority group that contains similar actions to the group that you are creating. Otherwise, you can select **None** to deselect all actions. To see all actions that are in a particular authority group, highlight the authority group, and then select **Overview**.
4. Select **Update** from the system pull-down menu to save your changes and exit from the notebook.

12.2 Creating or Modifying Interest Groups

This section provides instructions for manually editing the *interest.ld* file to create or modify interest groups. When you change the *interest.ld* file, you must also reload the contents of the interest table.

Subtopics

- 12.2.1 Editing the *interest.ld* File
- 12.2.2 Reloading the Interest Table
- 12.2.3 Using the GUI

12.2.1 Editing the interest.ld File

To add new interest groups or add actions to an existing interest group, edit the *interest.ld* file from the server machine in the directory where the family's database exists. Add entries to the file, using the following format:

```
InterestGroup|ActionName
```

InterestGroup

This is the name of an existing interest group or the name of a group that you are creating. The name can be up to 15 characters; it cannot contain blanks, tabs, or vertical separators. For an existing interest group, type the name exactly as it appears in the database table. The default names provided by IBM use all lowercase characters.

ActionName

This is the name of an existing TeamConnection action. Specify only one action per entry. You must type the name exactly as it appears in the database table. Certain actions cannot be included in an interest group (see the *IBM TeamConnection for OS/2 Version 1 User's Guide*).

12.2.2 Reloading the Interest Table

Whenever you change the *interest.ld* file, you must reload the contents of the interest table before your users can use the new and changed interest groups. You can reload the interest table as often as necessary. We recommend that you stop the family server before you reload the interest table.

To reload the interest table, issue the following command from the server machine in the directory where the *interest.ld* file is stored:

```
fhclintr interest.ld dbpath\familyname
```

where:

- *dbpath* is the name of the directory where your family's database exists.
- *familyname* is the name of your TeamConnection family.

Before issuing this command, ensure that the TC_FAMILY environment variable is set to the correct family name.

To verify that the interest table loaded correctly, use the *report -view* command to generate a report on the interest table. For example, to verify that a new interest group named *general* was added to the table, type the following from a command line on a client machine:

```
teamc report -view interest -where "name='general'"
```

If the interest table loaded correctly, information about the *general* interest group appears. If the interest table did not load correctly, make the necessary changes to the *interest.ld* file and run the command again.

For more information about the *report -view* command, refer to the *IBM TeamConnection for OS/2 Commands Reference*.

12.2.3 Using the GUI

Use the **Interest Groups** page (see "The Interest Groups Page" in topic 5.3) to associate such actions as *new*, *rename*, and *delete* with interest groups.

12.3 Configuring Component or Release Processes

In this section we provide instructions for using the GUI and manually editing the *comproc.ld* and *relproc.ld* files to configure processes. When you change the *.ld* files, you must also reload the contents of the configurable process tables.

A Component Process determines the amount of planning and designing required before work begins on a defect or feature written against the component. This is also where you specify whether the originator is required to verify that the work was done correctly.

TeamConnection is shipped with several predefined processes for components. If these processes do not meet the needs of your development organization, you can create your own processes by combining some of the predefined subprocesses that IBM provides.

Release processes determine the amount of tracking applied to part changes and the procedure for integrating those changes into build. They control the work involved in developing the parts, fixing defects, implementing features, and building the product.

TeamConnection is shipped with several predefined processes for releases. If these processes do not meet the needs of your development organization, you can create your own processes by combining some of the predefined subprocesses that IBM provides.

Subtopics

- 12.3.1 Editing the *comproc.ld* and *relproc.ld* Files
- 12.3.2 Reloading the Configurable Process Tables
- 12.3.3 Using the GUI for Component Processes
- 12.3.4 Using the GUI for Release Processes

12.3.1 Editing the *comproc.ld* and *relproc.ld* Files

Information about configurable processes for components is stored in the *comproc.ld* file. Information about configurable processes for releases is stored in the *relproc.ld* file. When the family is created, the configurable process tables are created, based on the settings in the *comproc.ld* and *relproc.ld* files. If you modify the configurable process tables after the family is created, edit the *.ld* files and then run the *fhclproc* command.

To add new processes or change existing processes, edit the *comproc.ld* file for component processes or the *relproc.ld* file for release processes. These files are in the directory created for the family's database. Add entries to the file, using the following format:

```
ProcessName|SubprocessName
```

ProcessName

The name of the process you are creating. The name can be up to 15 characters long; it cannot contain blanks, tabs, or vertical separators.

SubprocessName

The name of a TeamConnection subprocess. You can specify only one of the following subprocesses for each entry:

For components:

- ☐ None
- ☐ dsrDefect
- ☐ dsrFeature
- ☐ verifyDefect
- ☐ verifyFeature

For releases:

- ☐ None
- ☐ Approval
- ☐ Fix
- ☐ Driver
- ☐ Test
- ☐ Track

If you want to include more than one subprocess, you must have an entry for each subprocess. Type the name exactly as it appears in the database.

12.3.2 Reloading the Configurable Process Tables

After you edit a *.ld* file, use the *fhclproc* command to reload the contents of the configurable component or release process tables with the changed values. Before issuing this command, ensure that the *TC_FAMILY* environment variable is set to the correct family name. The format of the *fhclproc* command when reloading the component process table is:

```
fhclproc comproc.ld dbpath\familyname c
```

The format of the *fhclproc* command when reloading the release process table is:

```
fhclproc relproc.ld dbpath\familyname r
```

where:

- ☐ *dbpath* is the name of the directory where your family's database exists.
- ☐ *familyname* is the name of the TeamConnection family.
- ☐ **c** indicates that you are reloading the component process table.
- ☐ **r** indicates that you are reloading the release process table.

To verify that the command successfully modified the tables, use the *report -view* command to generate a report. To do this, type one of the following commands from an OS/2 or TeamConnection command line:

```
teamc report -view Cfgcomproc  
teamc report -view Cfgrelproc
```

If the table did not load correctly, make the necessary changes to the *comproc.ld* or *relproc.ld* file and run the command again. For more information about the *report -view* command, refer to the *IBM TeamConnection for OS/2 Commands Reference*.

12.3.3 Using the GUI for Component Processes

Use the **Component Processes** page (see Figure 30 in topic 5.5.2) to configure the component processes. On the **Component Processes** page you can modify existing or define new component processes by including any of the existing subprocesses.

12.3.4 Using the GUI for Release Processes

Use the **Release Processes** page (see Figure 31 in topic 5.6.2) to configure the release processes. On the **Release Processes** page you can modify existing or define new release processes by including any of the existing subprocesses.

12.4 Defining Configurable Field Types

You, the family administrator, can configure fields so TeamConnection will more closely match your development environment. By changing existing fields in the defect and feature tables and by creating new fields for the defect, feature, part, and user tables, your development group can store information that is customized to your environment and terminology. For example, you might want to add a field called *PubImpact* to the defect and feature tables. Programmers can then use this field to notify the writing team as to whether or not a defect or feature affects the accuracy of the documentation.

Before you configure new fields, you must decide whether you are going to give users a selection list of field values or let them enter free-form text. To do this, you specify a field type when you create the field. For example, if you create a *PubImpact* field, you might want to create a new field type called *PubImpact* (yes, it can have the same name as your field). If you assign the attributes of *yes*, *no*, and *maybe* to this field, writers can access the user interface or issue a command to get a list of all defects or features that affect the publications. If you also add an attribute of *done*, the writers can indicate when they have finished updating the documentation.

Default configurable fields are shipped by IBM and are installed when the TeamConnection server is configured. If you do not want to use these defaults, you can change them at any time after the server is configured.

Subtopics

12.4.1 Manually Defining Configurable Field Types

12.4.2 Creating and Modifying Configurable Fields Using the *chfield* Command

12.4.3 Working with Configurable Fields Using the GUI

12.4.1 Manually Defining Configurable Field Types

In this section we provide instructions for manually editing the *config.ld* file to define configurable field types. When you change the *config.ld* file, you must also reload the contents of the config table. The *config.ld* file is in the directory where the family's database is installed. When adding entries to the file, follow the existing format of the file:

```
fieldType|value|default|0|0|description
```

Information about configurable field types is stored in the config table. After you modify the config table, you must reload it (see "Reloading the Config Table" in topic 12.4.1.1). The config table consists of the following information:

Field type

Identifies the types of configurable fields that are defined for your family. You specify one of these types when you configure a new field. You can create new types, and you can configure the acceptable values for each type. You must have at least one value for each type. The type field can have up to 15 characters, but it cannot contain blank spaces or tabs.

The following describes the configurable field types that are shipped by IBM:

priority

An indication of the timing or scheduling requirements for resolving a defect or feature

drivertype

The type of driver in which the defect or feature resolution should be included

severity

An indication of how severe a defect is

answerAccept

The answer to a defect or feature that the owner uses when accepting it

answerReturn

The answer to a defect or feature that the owner uses when returning it

defectPrefix

A prefix indicating the type of defect. The prefix attribute can distinguish a defect from a feature when a user looks at information regarding both. Use unique prefixes for defects.

featurePrefix

A prefix indicating the type of feature. The prefix attribute can distinguish a defect from a feature when a user looks at information regarding both. Use unique prefixes for features.

phase

The current stage in development when the defect was discovered or injected into the code

symptom

An indication of the problem.

Note: If the default configurable fields shipped by IBM are not installed, the *priority*, *phase*, and *symptom* types are not used.

Value

This field represents the choices the user has for the configurable field. You can add choices to the default fields shipped by IBM and to the fields created specifically for your family. The value can have up to 15 characters, but it cannot contain spaces or tabs.

Note: Because from the command line a user can abbreviate these values, you cannot define a value that can be an abbreviation of another value of the same type. For example, you cannot add a value of build to the phase type, because a value of building already exists, or, if a value of 1 exists for the severity type, you cannot add a severity value of 12.

Default

This field indicates whether the defined name is used as the default when the user does not enter a value for the configuration type. Valid values are either *yes* or *no*, and only one name for each configuration type can have the default field set to *yes*. The config table shipped by IBM does not have any of the values set as defaults.

Value1 and Value2

Keep these fields set to 0. They are reserved for future use.

Description

This field contains the description of each value. The description field cannot contain more than 63 characters, but it can be set to blank. The description with the defined values appears on the GUI window when the field is displayed.

The default configuration field types, along with their attributes, that IBM ships are listed in the *IBM TeamConnection for OS/2 Version 1 User's Guide*.

Subtopics

12.4.1.1 Reloading the Config Table

12.4.1.1 Reloading the Config Table

When you edit the *config.ld* file and change any values, you must reload the contents of the config table so that TeamConnection recognizes the changes. You can reload the config table as often as necessary. We recommend that you stop the family server before you reload the table. To reload the config table, issue the following command from the server machine in the directory where the *config.ld* file is stored:

```
fhclcnfg config.ld dbpath\familyname
```

where:

- ☐ *dbpath* is the name of the directory where your family's database exists.
- ☐ *familyname* is the name of your TeamConnection family.

Before issuing this command, ensure that the TC_FAMILY environment variable is set to the correct family name.

Changing values in the config table does not change any values that are already in the database for existing records. To verify that the command successfully modified the config table, use the *report -view* command to generate a report. Type the following from an OS/2 or TeamConnection command line:

```
teamc report -view config
```

If the config table did not load correctly, make the necessary changes to the *config.ld* file and run the command again. For more information about the *report -view* command, refer to *IBM TeamConnection for OS/2 Version 1 User's Guide*.

12.4.2 Creating and Modifying Configurable Fields Using the `chfield` Command

This section provides instructions for using the **chfield** command to create and modify configurable fields.

From the server machine, use the `chfield` command to:

- ☐ Change the properties of existing configurable fields
- ☐ Create new configurable fields
- ☐ Copy configurable fields from another family
- ☐ Create report formats for new fields

The following is the syntax for the `chfield` command:

```
chfield -object name [-source name ]
```

where:

- ☐ *object name* is the object to configure. Valid values are *defect*, *feature*, *part*, or *user*.
- ☐ *source name* is the name of a system-generated file containing configurable fields. Valid values are *default* or an existing family name. The default family is the value of the `TC_FAMILY` environment variable.

If you specify an existing family name that is not fully qualified, the directory path of the current TeamConnection family is used. This is the path specified in the `TC_DBPATH` environment variable in your `CONFIG.SYS` file.

Type the following command to display the configurable fields for the specified object (*defect*, *feature*, *part*, or *user*) in the current family:

```
chfield object
```

When you issue the `chfield` command, system files residing in the directory where TeamConnection is installed are either generated or updated. The new fields do not take effect until the server is started again. The files are:

For the defect object:

- ☐ `cfgfield\defect.tbl`
- ☐ `cfgfield\defect.fmt`

For the feature object:

- ☐ `cfgfield\feature.tbl`
- ☐ `cfgfield\feature.fmt`

For the part object:

- ☐ `cfgfield\part.tbl`
- ☐ `cfgfield\part.fmt`

For the user object:

- ☐ `cfgfield\user.tbl`
- ☐ `cfgfield\user.fmt`

Note:

1. Do not manually modify these files; the `chfield` command does that for you. Manual modification can cause incorrect values to appear in your configurable fields.
2. The `part.tbl` and `user.tbl` files do not exist until they are generated by the `chfield` program.

Subtopics

12.4.2.1 Creating Configurable Fields

12.4.2.2 Creating Configurable Fields by Copying from Another Family

12.4.2.3 Updating Configurable Fields

12.4.2.1 Creating Configurable Fields

Follow these steps to create a configurable field:

1. Type the *chfield* command, specifying the object to which you are adding the field (for example, **chfield -object defect**). You are prompted to select an option.
2. Select option **1**, Create a configurable field in the report and table file. You are prompted for the following information (you must type information at each prompt):

Enter DB Column Name

This is an informational tag that will appear in the *.fmt* file. It serves as an index.

Is field active

This flag indicates whether the field is active. A value of *yes* means the field is active; *no* means it is inactive. An inactive field is ignored, and its location in the database is replaced by the next field entry in the *.tbl* file.

Enter CMD Attribute

This name is used as the command attribute on the command line. For example, the attribute for the *PubImpact* field on the defect command might be *-pubImpact*.

Enter Field Label

This name appears in the user interface window as the field name.

Enter Title

This name appears in the header title for the object window--for example, *PublicationImpact* . Spaces are not allowed in the title. If this name is not provided, the configurable field is not displayed in the object window.

Is it an attribute for "create/open" action

This flag shows whether the field is one of the attributes within the *part create*, *user create*, *defect open*, or *feature open* actions. Valid values are *yes* and *no*.

Is it a required field for "create/open" action

This flag indicates whether the field is required for the *create* or *open* actions. Valid values are *yes* and *no*.

Enter the field type

This value matches one of the configurable field types defined in the *config.ld* file. See the *IBM TeamConnection for OS/2 Version 1 User's Guide* for a listing of the configuration field types that are shipped by IBM. By specifying a type, you limit the acceptable values that a user can type in the field.

Enter the width (in bytes) available for formatting

The maximum allowable width of the value that can be entered in the field. This can be a number of 1 to 25.

When you exit the display, the file is changed, but the database is not altered until the server is stopped and then restarted. Figure 176 shows an example of the display after typing information for creating a new field called *Developer*.

```
+-----+
|
| You have selected to create a new field
| Limit 15 characters per entry
|
| Enter DB Column Name :
| developer
| Is field active {Y/N} :
| Y
| Enter CMD Attribute :
| developer
| Enter Field Label :
| Developer:
| Enter Title :
| Developer
| Is it an attribute for "create/open" action {Y/N} :
| Y
| Is it a required field for "create/open" action {Y/N} :
| N
| Enter the field type :
|
+-----+
```

:	developer	:
:	Enter width (in bytes) available for formatting. Acceptable range of	values: 1 to 25
:	10	:
+-----+		

Figure 176. The chfield Display with Configurable Field Information

Creating Configurable Fields by Copying from Another Family

12.4.2.2 Creating Configurable Fields by Copying from Another Family

You can copy configurable field information from an existing family. Follow these steps:

1. If the source family is on a different machine from the target family machine, copy the *.fmt* and *.tbl* system files for each object that you want to copy from the source to the target machine.
2. Type the following command to create the fields for the specified object:

```
chfield -object objectName -source TargetFamilyName
```

3. Restart the family server so that TeamConnection recognizes the new fields.

To see the current field information, use the appropriate object command and the *-configInfo* action flag. For example, to see the field information for the user table in family *rdev*, type the following command:

```
teamc user -configInfo -family rdev
```

12.4.2.3 Updating Configurable Fields

Follow these steps to update a configurable field:

1. Type the *chfield* command, specifying the object in which you are updating the field (for example, *chfield -object defect*). You are prompted to select an option.
2. Select option 2, Update a configurable field in the report and table file. The current information for that object is displayed.
3. Type the database column name you are updating.
4. For each prompt, either change the value or press Enter to keep the same value. "Editing the userExit File" in topic 12.6.1 describes each prompt.

12.4.3 Working with Configurable Fields Using the GUI

When you add fields, TeamConnection displays them on the GUI. However, the help information for the GUI and the commands will not reflect your new or changed fields.

IBM ships configurable field types that have defined values. You can use the default field types as is or change their attributes. For example, IBM defines a *Severity* field. This field has valid values of 1, 2, 3, and 4. You could add an additional value of 5, or you could change the description of what value 2 represents. You can also create new configurable field types. This allows you to structure problem tracking information for your development environment.

In the sections that follow we discuss:

- ☐ Creating and modifying configurable fields
- ☐ Working with the **Defect** pages
- ☐ Working with the **Feature** pages
- ☐ Working with the **Part** pages
- ☐ Working with the **User** pages

Subtopics

- 12.4.3.1 Creating and Modifying Configurable Fields
- 12.4.3.2 Working with the Defect Pages
- 12.4.3.3 Working with the Feature Pages
- 12.4.3.4 Working with the Part Pages
- 12.4.3.5 Working with the User Pages

12.4.3.1 Creating and Modifying Configurable Fields

Follow these steps to create or modify configurable fields from the Family Administrator GUI:

1. Display the family icon's pop-up menu, then select **Open > Settings**. The **Settings** notebook appears.
2. Select the **Configurable Fields** page (see Figure 29 in topic 5.4.2) to display the current settings. This page has four tabs: **Defects**, **Features**, **Parts**, and **Users**. Select one of the tabs to display the configurable field settings for the selected object. From this page, you can:
 - ☐ Change information about a field. Highlight the field from the **Configurable Fields** list. Information about the selected field is displayed to the right of the list, and you can then change this information.
 - ☐ Create a new configurable field. Type the new name in the entry field, and then select **New** to add the field type to the list. Select or type the appropriate information to the right of the list.
 - ☐ Rename an existing field. Type the name in the entry field and then select **Rename**. If you want to change the possible values for the field, type the appropriate information to the right of the list.
 - ☐ Change the report format for the field. Select the right arrow at the bottom right corner of the notebook page. As noted earlier, you are limited to the number of configurable fields that you can have. You cannot delete configurable fields, but you can rename them and change their properties.
3. Select **Close** from the system pull-down menu to save your changes and exit from the notebook.

12.4.3.2 Working with the Defect Pages

There are three pages for defining the defect configurable fields. You can use:

- ☐ Page one to set up and define the field types
- ☐ Page two to create and modify configurable fields
- ☐ Page three to change report formats by specifying which fields TeamConnection includes in the report, which fields TeamConnection does not include in the report, and the width of the columns TeamConnection uses to display the fields.

The **Defect** page has the following fields and push buttons:

Configurable Fields for Defects

The list of existing configurable fields for defects

Field Type

The type of configurable fields that are defined for your family. You must have at least one value for each type. The field can have up to 15 characters but cannot contain blank spaces or tabs.

Attribute

The command attribute on the command line. For example, the attribute for the *PubImpact* field on the *Defect* command might be *-pubImpact*.

Field Label

The field name that appears in the GUI window

Title

The name that appears in the header title for the object window, for example, *PublicationImpact*. Spaces are not allowed in the title. If a name is not provided, the configurable field is not displayed in the object window.

Active

The field is available to the user.

Required

Data must be entered in the field by the user.

Allowed on Create/Open

The user can enter information in the field on Create/Open.

New

Creates a new configurable field

Rename

Renames the selected configurable field

Undo

Removes any current changes and returns the fields to their previous values

Help

Displays help information about the window

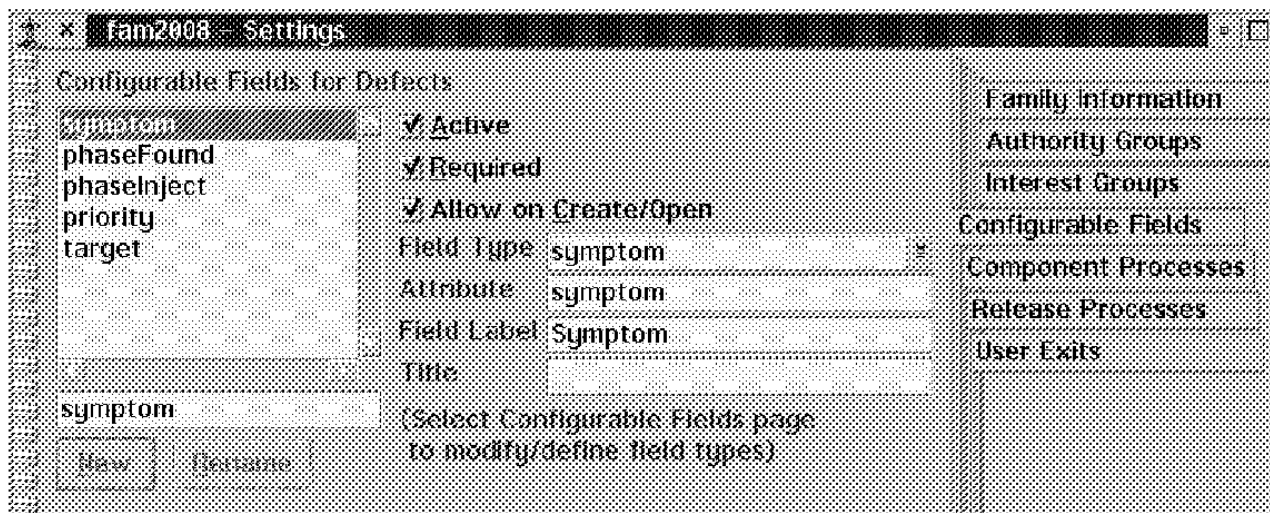


Figure 177. Configurable Fields Defect Page

12.4.3.3 Working with the Feature Pages

There are three pages for defining the feature configurable fields. You can use:

- ☐ Page one to set up and define the field types
- ☐ Page two to create and modify configurable fields
- ☐ Page three to change report formats by specifying which fields TeamConnection includes in the report, which fields TeamConnection does not include in the report, and the width of the columns TeamConnection uses to display the fields.

The **Feature** page has the following fields and push buttons:

Configurable Fields for Features

The list of existing configurable fields for features

Field Type

The type of configurable fields that are defined for your family. You must have at least one value for each type. The field can have up to 15 characters but cannot contain blank spaces or tabs.

Attribute

The command attribute on the command line. For example, the attribute for the *PubImpact* field on the *Feature* command might be *-pubImpact*.

Field Label

The field name that appears in the GUI window.

Title

The name that appears in the header title for the object window, for example, *PublicationImpact*. Spaces are not allowed in the title. If a name is not provided, the configurable field is not displayed in the object window.

Active

The field is available to the user.

Required

Data must be entered in the field by the user.

Allowed on Create/Open

The user can enter information in the field on Create/Open.

New

Creates a new configurable field

Rename

Renames the selected configurable field

Undo

Removes any current changes and returns the fields to their previous values

Help

Displays help information about the window

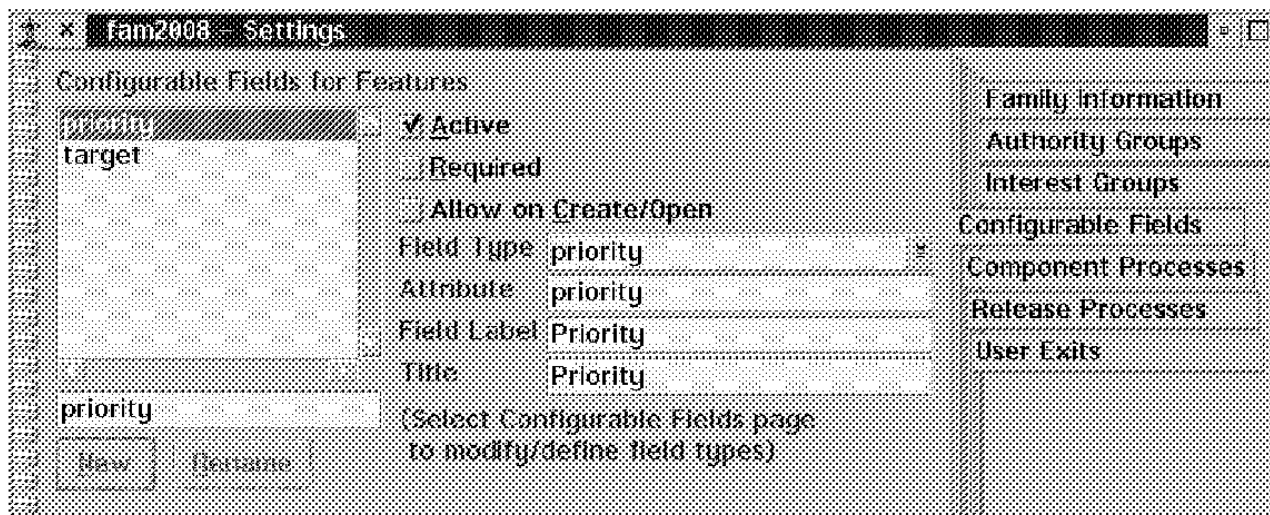


Figure 178. Configurable Fields Feature Page

12.4.3.4 Working with the Part Pages

Use the **Part** page (see Figure 179) to set up and define the part configurable fields. You can use:

- ☐ Page one to define field types
- ☐ Page two to create and modify configurable fields
- ☐ Page three to change report formats by specifying which fields TeamConnection includes in the report, which fields TeamConnection does not include in the report, and the width of the columns TeamConnection uses to display the fields.

The **Part** page has the following fields and push buttons:

Configurable Fields for Parts

The list of existing configurable fields for parts

Field Type

The type of configurable fields that are defined for your family. You must have at least one value for each type. The field can have up to 15 characters but cannot contain blank spaces or tabs.

Attribute

The command attribute on the command line. For example, the attribute for the *PubImpact* field on the *Part* command might be *-pubImpact*.

Field Label

The field name that appears in the GUI window

Title

The name that appears in the header title for the object window, for example, *PublicationImpact*. Spaces are not allowed in the title. If a name is not provided, the configurable field is not displayed in the object window.

Active

The field is available to the user.

Required

Data must be entered in the field by the user.

Allowed on Create/Open

The user can enter information in the field on Create/Open.

New

Creates a new configurable field

Rename

Renames the selected configurable field

Undo

Removes any current changes and returns the fields to their previous values

Help

Displays help information about the window

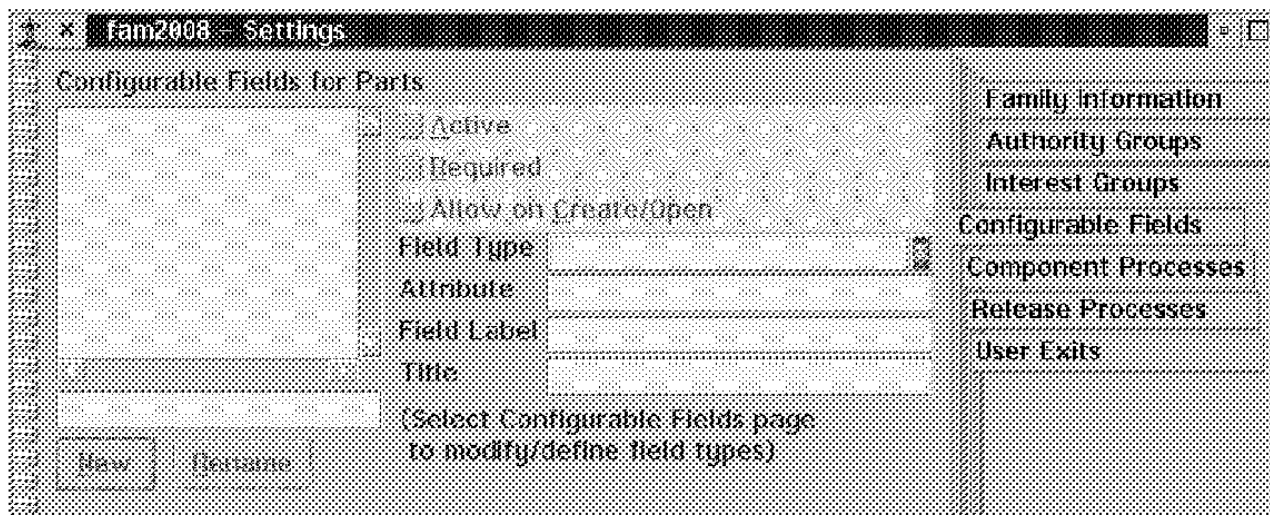


Figure 179. Configurable Fields Part Page

12.4.3.5 Working with the User Pages

Use the **User** page (see Figure 180) to set up and define the user configurable fields. You can use:

- ☐ Page one to define field types
- ☐ Page two to create and modify configurable fields
- ☐ Page three to change report formats by specifying which fields TeamConnection includes in the report, which fields TeamConnection does not include in the report, and the width of the columns TeamConnection uses to display the fields.

The **User** page has the following fields and push buttons:

Configurable Fields for Users

The list of existing configurable fields for users

Field Type

The type of configurable fields that are defined for your family. You must have at least one value for each type. The field can have up to 15 characters but cannot contain blank spaces or tabs.

Attribute

The command attribute on the command line. For example, the attribute for the *PubImpact* field on the *User* command might be *-pubImpact*.

Field Label

The field name that appears in the GUI window

Title

The name that appears in the header title for the object window, for example, *PublicationImpact*. Spaces are not allowed in the title. If a name is not provided, the configurable field is not displayed in the object window.

Active

The field is available to the user.

Required

Data must be entered in the field by the user.

Allowed on Create/Open

The user can enter information in the field on Create/Open.

New

Creates a new configurable field

Rename

Renames the selected configurable field

Undo

Removes any current changes and returns the fields to their previous values

Help

Displays help information about the window

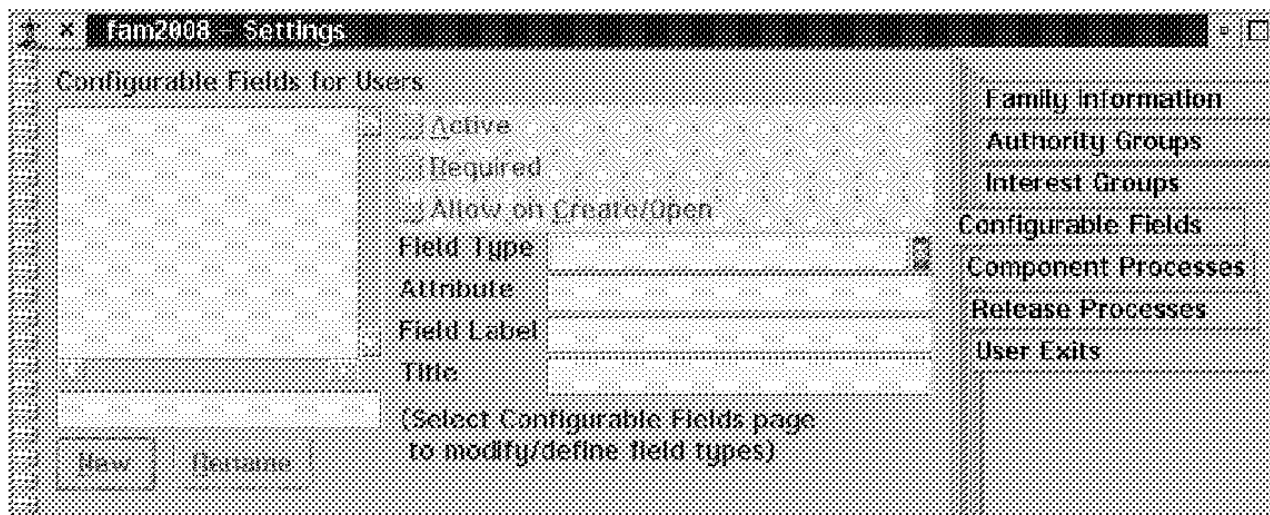


Figure 180. Configurable Fields User Page

12.5 Changing Report Formats

In the sections that follow we explain how you can both manually and through the GUI change the position of report fields on the reports TeamConnection generates for the *defect*, *feature*, *part*, and *user* objects.

Subtopics

12.5.1 Manually Changing Report Formats

12.5.2 Changing Report Formats by Using the GUI

12.5.1 Manually Changing Report Formats

You can use the system editor to edit the following files:

- ☐ cfgfield\defect.fmt
- ☐ cfgfield\feature.fmt
- ☐ cfgfield\part.fmt
- ☐ cfgfield\user.fmt

Before you change the report formats, you might want to make backup copies of these files.

Each .fmt file is divided into the following five sections, separated by colons:

- ☐ StanzaViewFormat
- ☐ StanzaViewColumn
- ☐ TableViewFormat
- ☐ TableViewColumn
- ☐ TableViewHeader

The column sections describe the column name of each of the labels specified in the format sections. The header section specifies how the columns appear in the table format. The format sections specify the layout of the report. For example, a format specification of **%3\$-25.25s** indicates the following:

```
%
    Start of format specification

3
    The sequence number of the field that is generated by TeamConnection.
    The dollar sign must appear after the sequence number.

-
    The output is left-justified.  If you do not include this character,
    the output is right-justified.

25
    The minimum number of characters (bytes) of output

.25
    The maximum number of characters (bytes) printed for all or part of
    the output field, or minimum number of digits printed for integer
    values

    If you do not want the field displayed, type 0.0.  For example, you
    have three sequence fields: 1, 2, and 3.  If you do not want sequence
    2 displayed, you type:

        %1$-4.4s %2$-0.0s %3$-15.15s

s
    Type of data:

    s    for strings

    ld   for integers
```

You can specify only a data type of **s** for configurable fields. Use **ld** to display existing values, such as defect age.

You can also change or delete the format specification. Before you change a format specification, be aware of the following:

- ☐ A format specification in the stanza view does not have to match the format specification for the same field in the table view.
- ☐ Information in a stanza report appears in columns. When you specify the identical minimum and maximum number of characters for all fields appearing in a column, the report columns are left-justified. For example, Figure 181 shows all the fields in the first column defined as 25.25.
- ☐ When you change a format specification in the table view, adjust the matching heading length in the table view header section. Otherwise, information will not appear correctly under the headings when users display the table.

Figure 181 shows a sample report format for the defect table after configurable fields have been added. The changes are noted in bold font and are described following the figure.

+-----+

```

# StanzaViewFormat

prefix      %1$s
name        %2$s
reference    %21$s
abstract     %9$s
duplicate    %13$s

state       %6$-25.25s  priority    %28$-20.20s
severity    %8$-25.25s  target      %29$-20.20s
age         %10$1d

compName    %3$-25.25s  answer      %7$-20.20s
release     %4$-25.25s  symptom     %25$-20.20s
envName     %11$-25.25s phaseFound  %26$-20.20s
level       %12$-25.25s phaseInject  %27$-20.20s

addDate     %15$-25.25s assignDate   %16$-20.20s
lastUpdate  %14$-25.25s responseDate %17$-20.20s
endDate     %18$-25.25s

ownerLogin  %5$-25.25s  originLogin  %22$-20.20s
ownerName   %19$-25.25s originName   %23$-20.20s
ownerArea   %20$-25.25s originArea   %24$-20.20s

developer    %30$-25.25s

:
# StanzaViewColumn
prefix,name,reference,abstract,duplicate,state,priority,severity,target,age,compName,answer,
responseDate,endDate,ownerLogin,originLogin,ownerName,originName,ownerArea,originArea,developer
:
# TableViewFormat
%1$-4.4s %2$-15.15s %3$-15.15s %6$-8.8s %22$-8.8s %5$-8.8s %8$-3.3s %10$-3.31d %28$-4.4s %9$-55
.55s %4$-0.0s %7$-0.0s %11$-0.0s %12$-0.0s %13$-0.0s %14$-0.0s %15$-0.0s %16$-0.0s %17$-0.0s %18$-0.0s %19$-0.0s %20$-0.0s %21$-0.0s %23$-0.0s %24$-0.0s %25$-0.0s %26$-0.0s %27$-0.0s %30$-9.9s
:
# TableViewColumn
prefix,name,compName,state,originLogin,ownerLogin,severity,age,priority,abstract,developer
:
# TableViewHeader
pref name      compName      state      originLo ownerLog sev age prio abstract developer
-----
:

```

Figure 181. Sample Report Format after Adding Configurable Fields

In Figure 181, the format of the defect report shipped by IBM was modified as follows:

- ☐ A new label, *developer*, was added at the end of the *StanzaViewFormat* section, and the format specification **%30\$-25.25s**.
- ☐ The column name, *developer*, was added as the last entry in the *StanzaViewColumn* section.

Note: When you edit the *StanzaViewColumn*, you must maintain a continuous line of text. Control characters are ignored and appear as output in the report.

- ☐ **%30\$-9.9s** was added in the corresponding position for the *developer* entry in the *TableViewFormat* section.
- ☐ The column name, *developer*, was added in the *TableViewColumn* section.
- ☐ A new label, *developer*, was added in the *TableViewHeader* section and the corresponding dashes were added in the next line.

12.5.2 Changing Report Formats by Using the GUI

TeamConnection users can view information about an object. You can choose the field information you want to present to the user and where on the report the information appears. When you create a field, TeamConnection updates the report format by adding the new field to the report. You can also change how this information is viewed. Before you change the report formats, you might want to make backup copies of the .fmt files.

To change the position of a field in the stanza view format, or to change or delete the format specification for a field, follow these steps:

1. From the **Defect**, **Feature**, **Part**, or **User** page, use the right arrow at the bottom-right corner of the notebook page to access the **format** page (see Figure 177 in topic 12.4.3.2, Figure 178 in topic 12.4.3.3, Figure 179 in topic 12.4.3.4, or Figure 180 in topic 12.4.3.5). Select the arrow once for the **Stanza View Format** page (see Figure 182).
2. Select a field in the stanza view (a field is shown in black and not contained in << >>).
3. If you want to add a new field, select either **Insert Before** or **Insert After** to insert a line either before or after the selected item.
4. Type the label for the field.
5. Select the area for the field value (the field value is shown in blue and contained in << >>). If you are creating a new field, a small box appears where the new field value will be placed.
6. Click mouse button 2 to view a list of possible values for the field.
7. Select a value from the list.

To delete a line in the format, follow these steps:

1. Select either a field or a field value.
2. Select **Delete**.

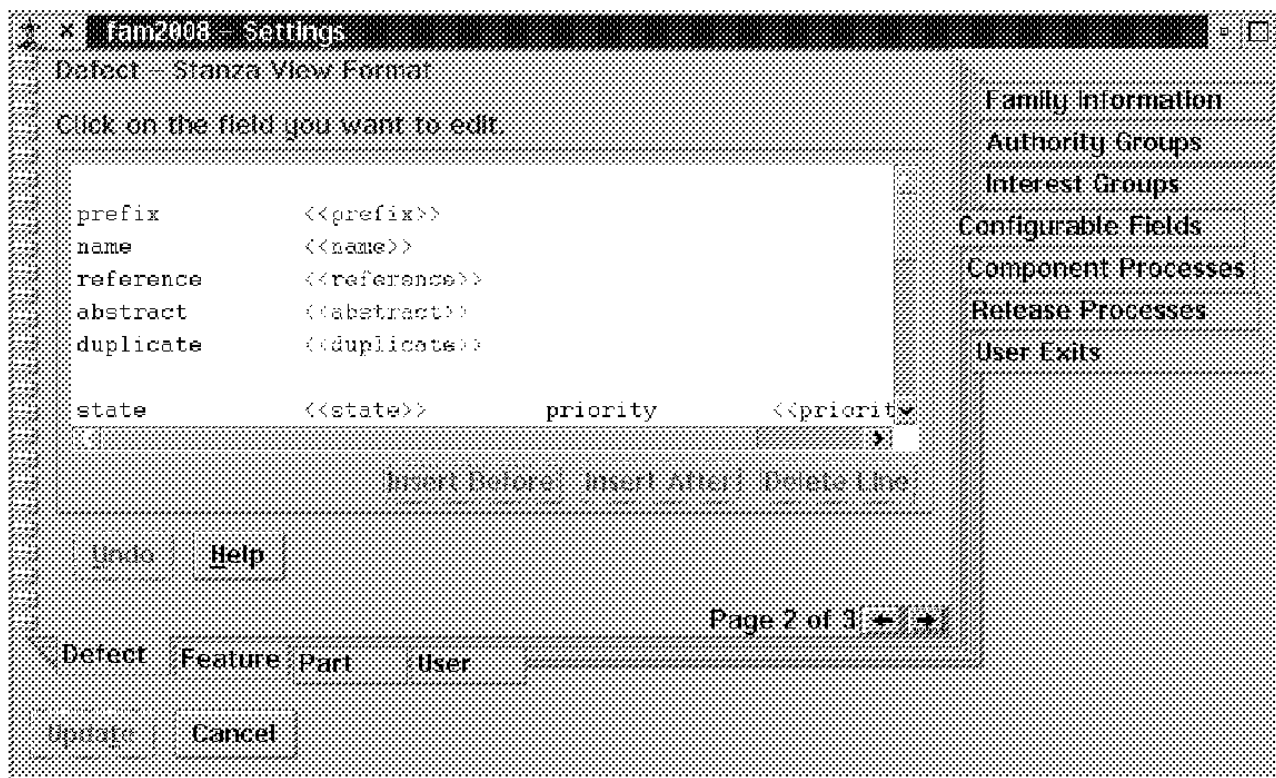


Figure 182. Stanza View Format Page

To change the position of a column in the table view format, or to add or remove a column, follow these steps:

1. From the **Defect**, **Feature**, **Part**, or **User** page (see Figure 177 in topic 12.4.3.2, Figure 178 in topic 12.4.3.3, Figure 179 in topic 12.4.3.4, or Figure 180 in topic 12.4.3.5), use the right arrow at the bottom-right corner of the notebook page to access the **Table View Format** page (see Figure 183). Select the arrow twice for the table report format.
2. Select the item you want to add to or remove from the format.
3. Use the arrow icons located between the *Columns displayed* field and the *Columns not displayed* field to move the selected item.

To change the order in which the columns are displayed, follow these steps:

1. From the **Defect**, **Feature**, **Part**, or **User** page (see Figure 177 in topic 12.4.3.2, Figure 178 in topic 12.4.3.3, Figure 179 in topic 12.4.3.4, or Figure 180 in topic 12.4.3.5), select the right arrow in the bottom right corner of the notebook page twice to access the **Table View Format** page (see Figure 183).
2. Select the column you want to move.
3. Select the icons at the left to move the column either up or down the list.

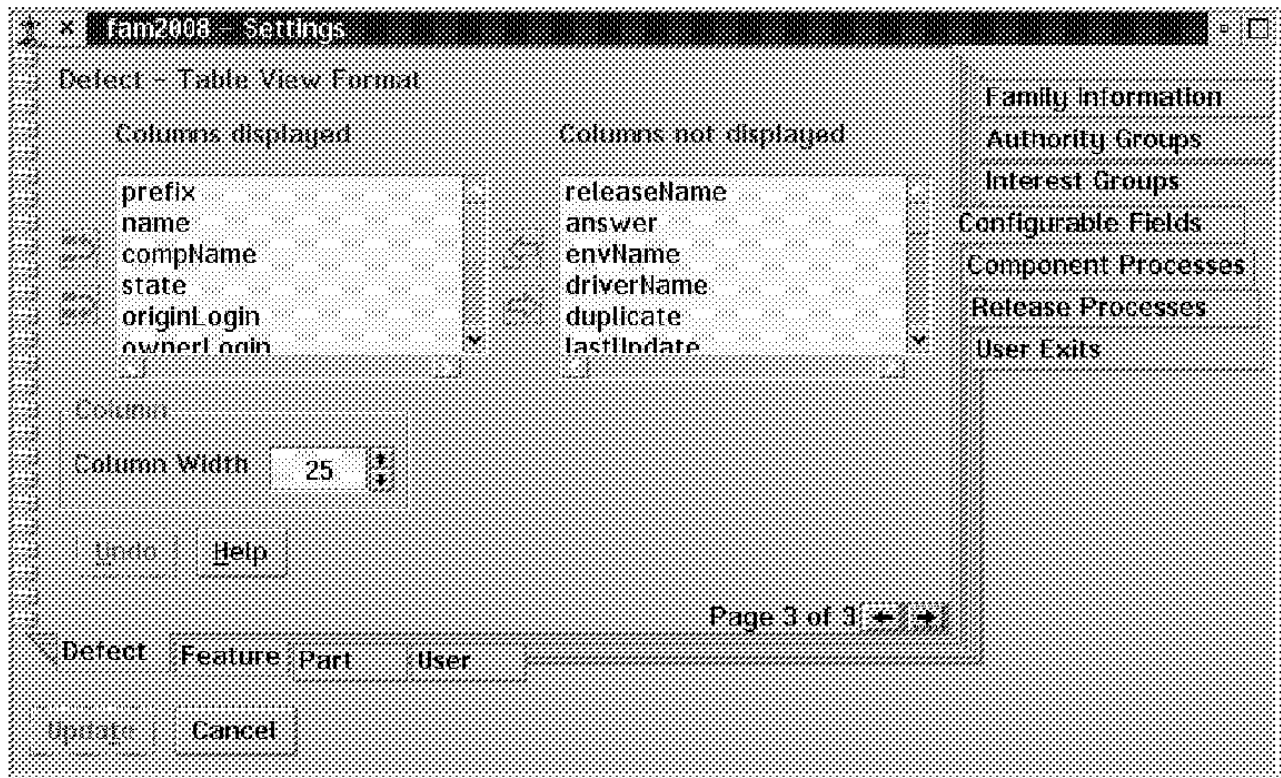


Figure 183. Table View Format Page

Note: The **Stanza View Format** and the **Table View Format** pages are the same for both the **Defect**, **Feature**, **Part**, and **User** pages.

12.6 Setting Up User Exits

In this section we provide instructions for manually updating the *userExit* file to add entries that call user-defined programs during the processing of TeamConnection actions.

Subtopics

12.6.1 Editing the userExit File

12.6.2 Using the GUI

12.6.1 Editing the userExit File

The *userExit* file has no defined actions until you add entries for the user exits that your organization will use. The entries you add specify the programs that you want started for specific TeamConnection actions. For each user exit, add an entry, using the following format:

```
Action  ExitID  UEprogram  UEparameters  #Comments
```

Use one or more blank spaces to separate each field in the entry. A line that begins with a # sign is a comment. You can have blank lines in the file.

The *userExit* file is located in the config subdirectory of the directory where your family's database is installed. Here is a description of each field in the entry:

Action

The name of the TeamConnection action that causes the user exit to start. You must type the name exactly as it appears in the database. See the list of actions in Appendix H, "Worksheets," in the *IBM TeamConnection for OS/2 Version 1 User's Guide* for the correct spelling and capitalization.

ExitID

Identifies when the user exit program is started during the course of the TeamConnection action. Valid values are 0, 1, 2, and 3 and indicate that the user exit program does the following:

- 0 Starts at the beginning of the TeamConnection action, before any initialization or access checking takes place
- 1 Starts after all TeamConnection checks are made and TeamConnection is ready to process the command
- 2 Starts after the TeamConnection action is completed. At this point, the action has been submitted to TeamConnection, and all database or library updates have been committed.
- 3 Starts when a previous user exit with an exit ID of 0 or 1 is not successful or the TeamConnection action is not successful. This exit ID allows the user exit program to clean up what the other user exit programs started.

UEprogram

The name of the user exit program. The program must exist in the path directory of your CONFIG.SYS file.

UEparameters

A variable-length list of character string parameters provided to the user exit program

#Comments

A comment about the user exit program. This field is optional.

TeamConnection does not recognize the updates to the *userExit* file until you stop and restart the TeamConnection server.

12.6.2 Using the GUI

Use the **User Exits** page (see Figure 32 in topic 5.7.2) to identify programs that should be executed when certain TeamConnection actions are performed.

13.0 Chapter 13. Software Configuration Management and Change Management

This chapter briefly describes the objectives of and advantages associated with the processes known as configuration management and change management. It also discusses the relationship between configuration management and change management and the relationship between them and other processes such as project management and software development methodologies.

The elements produced during the development of an application are created progressively, as new requirements are discovered and old ones are refined. One quickly loses track of the state of development of the application when individual elements were introduced. You can have the latest CASE tools and a highly skilled, well-managed development organization and be following a superior development methodology but still find that your "as-built" application does not work as it was designed, coded, tested, or documented.

It is possible that the application that worked so well in testing cannot be successfully re-created for delivery simply because some fixes to the code were not integrated in the final build of the application. It is also equally likely that some unauthorized fixes did manage to find their way into the application, creating a mismatch in interfaces, calls to nonexistent subroutines, or inappropriate access to data that no one can seem to explain. Problems of this sort represent failures in configuration management and change management.

As Figure 184 shows, simply having all of the right parts does not ensure a successful outcome in software development. It takes configuration management to ensure that the right parts are put together in the right manner, and change management to ensure that any changes to those parts or their relationships are well thought out and deliberately applied.

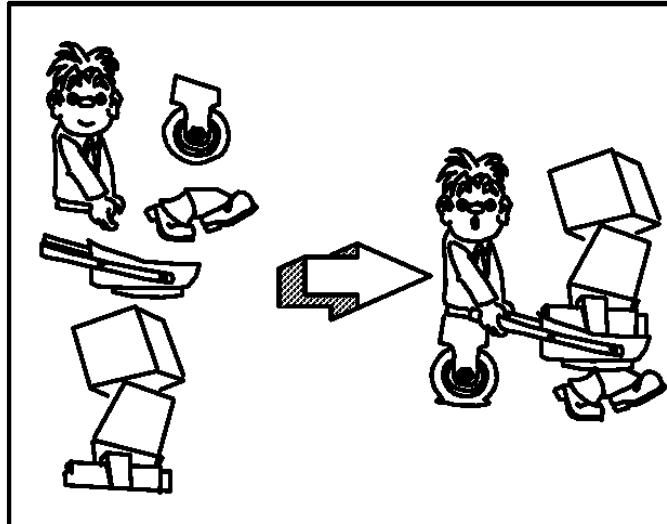
It is not enough to have:

excellent organization

a broken-in process

all necessary resources

**a wonderful set of
development tools**



to ensure you will produce a good system

Figure 184. Why Configuration Management and Change Management Are
Necessary

Subtopics

- 13.1 Why You Need Them
- 13.2 The Goals
- 13.3 The Formal Definition
- 13.4 What They Do
- 13.5 Who Needs Them
- 13.6 Interaction with Development Methodologies
- 13.7 Interaction with Project Management
- 13.8 Interaction with Quality Assurance
- 13.9 Configuration Management History and Statistics

13.1 Why You Need Them

Configuration management and change management are put into place to prevent or cure problems that contribute to high development and maintenance costs, missed schedules, and customer dissatisfactions such as these:

- ☐ Impossibility of re-creating an exact previous version that exhibits the failing characteristic reported by end users
- ☐ Inability to trace changes in the application source code back to the specific enhancement in the functional requirements, or to an approved design change, or to a specific reported failure in the previous version of the application
- ☐ Unnecessary reworks, inconsistent integration test results, and general confusion as to who is working on which problems
- ☐ Degradation of the application between one release and another because untested components were inadvertently incorporated into the whole
- ☐ Incompatibility between two pieces of software sharing a single interface that has changed in only one of them
- ☐ Impossibility of upgrading the application by incorporating a series of changes in a controlled and validated sequence
- ☐ Inability to generate different parallel versions of the same application because the parts that are unique to each version cannot be adequately identified
- ☐ Inability to report on the exact status of development or maintenance tasks
- ☐ Inability to identify exactly which developer is charged with implementing a particular change to the application

13.2 The Goals

The main purpose of configuration management and change management is to ensure consistency of the elements comprising the application, as well as consistency between the application and the documentation that defines it and supports it. Documentation that defines an application includes:

- ☐ Requirements specifications
- ☐ Interface specifications
- ☐ Design data, drawings, and specifications

Documentation that supports an application includes various end-user manuals and separately cataloged help information.

Another very important function is to identify precisely an application's significant development "baselines." These baselines are usually associated with a major project milestone event. Typical baselines include:

- ☐ Requirements analysis and specification
- ☐ Approved design documentation
- ☐ Evaluation prototype (alpha release)
- ☐ First integration test release (beta release)
- ☐ First end-user release
- ☐ Site-specific or platform-specific release

Identifying all changes to a given baseline and ensuring that they are incorporated into the next, newly forming baseline in an orderly and controlled manner are the core responsibilities of change management. Change control identifies and tracks problems and suggested enhancements to an application, thereby ensuring that each is carefully evaluated, and--if approved-- correctly implemented and incorporated into the application.

13.3 The Formal Definition

```
+-----+
| The classic definition of configuration management is given by the |
| U.S. Department of Defense in its standard on software development, |
| DOD-STD-2167A: |
| |
| "Configuration management is the discipline of identifying the |
| configuration of software systems at discrete points in time for the |
| purpose of controlling changes and maintaining traceability of changes |
| throughout the software system life cycle." |
+-----+
```

Configuration management is often divided into subprocesses:

- ☐ Configuration element identification: establishes a precise nomenclature for all configuration elements, such as files, subsystems, and systems, and provides a unique identifier for each of them
- ☐ Configuration control: manages the production of all elements of the application and the integration of those elements into a complete configuration
- ☐ Change control: records and tracks all change requests through their ultimate disposition
- ☐ Configuration audits: compares one baseline against its preceding baseline to ensure consistency between the two

13.4 What They Do

Changes to the application baselines occur continuously throughout the software life cycle. Configuration management and change management provide mechanisms to:

- ☐ Define and identify all elements of the application
- ☐ Define and identify how those elements are combined to build the complete application
- ☐ Keep track of change requests and their status (approved, assigned, implemented, tested, incorporated into the application)
- ☐ Maintain the traceability between all changes to the application and the change requests whose approval authorized those changes
- ☐ Retrieve a given previously established baseline
- ☐ Retrieve all changes applied between one baseline and the next baseline
- ☐ Allow gradual incorporation of sets of changes into the developing baseline and deincorporation of a set of changes that causes failure during integration testing

The processes associated with configuration management and change management ensure orderly development of software and enable a development history to be created. This history provides traceability and makes it possible to perform audits. It also enables statistics to be produced in order to evaluate the impact that an update may have on the software currently being developed or to be developed.

13.5 Who Needs Them

Any application development effort worth doing is worthy of configuration management and change management. This is usually self-evident in the case of big and medium-sized application development or software engineering efforts, but it is not always clear from the start of smaller efforts. Whether the need is from the first step of a project or better placed near the end of a project may depend on the size and complexity of the effort and the application. How complete and strict the procedures are that govern configuration management and change management may be determined by the size of the investment, the risk-to-benefit ratio, and company- or industry-imposed standards. But configuration management and change management are critical to the success of any major development effort, and they are cost-effective even during the maintenance phase of smaller efforts.

Subtopics

13.5.1 Big Development Efforts

13.5.2 Medium-Sized Development Efforts

13.5.3 Small Development Efforts

13.5.1 Big Development Efforts

Consider a large software development effort such as that of developing and maintaining a major operating system. With a reasonable assortment of application products and operating system code, a typical UNIX software system might take up 400 MB on disk when installed. It would support dozens of types of hardware peripherals, contain hundreds of end-user commands, include dozens of libraries and APIs, and provide a handful of compilers. At any one time, it probably has a half-dozen releases in the field and runs on at least three or four hardware architectures. The company selling such an operating system might employ hundreds of developers, testers, documenters, and quality assurance people to handle this job.

Such a company could not risk the loss of profit and sales caused by any confusion in the generation, maintenance, or delivery of its operating system and related software products. Communications among so many people could not be managed by word of mouth or random electronic communications. Project management would need sophisticated methods of measuring and auditing the development process. Quality assurance would have to be built in to every step of the development process. Clearly this company could not begin to manage an effort of this scope without very strict and widely encompassing configuration management and change management procedures and policies and significant automated tool support.

13.5.2 Medium-Sized Development Efforts

Now, consider a smaller business application development effort that handles customer, order, and payment information for products shipped on customer subscription. The application has COBOL, C, and C++ components, and some of the source code is automatically generated by a 4GL type of tool. It executes in parallel on two platforms, AIX and MVS, and will be implemented on AIX in two phases. It has a client/server architecture and is built on a commercial relational database. It has a nongraphical user interface on the MVS mainframe, and a GUI on AIX. It consists of a few dozen source files for each version on each platform, several build instruction files, some database instructions to create and populate the tables, and some text source files for end-user documents. This project will require only three or four developers to implement the AIX version and maintain the MVS version during that effort. It will have a project manager, some quality assurance oversight, and some end-user testing support.

This project will also require configuration management and change management. The few people involved in this project will not be able to modify and maintain multiple versions of the source files, readily identify the equivalent user interface files for each platform, and ensure that common code for both platforms is compatible with both compilers by keeping short-hand notes and sending each other occasional electronic mail. They will not be able to use simple mechanisms as separate directories and file naming conventions to ensure that use of common code is maximized while platform-specific and release-specific code is fetched properly during the product build process. In no time at all, this simple project will get out of hand.

13.5.3 Small Development Efforts

It is often necessary to convince software developers to apply configuration management and change management to smaller projects. The value of these disciplines to smaller development efforts is often underestimated. This type of management is perceived to require a level of effort and formality that is excessive compared to the total lines of code or the number of developers required to implement a small application. Developers also incorrectly associate configuration management and change management with a restraint on their creativity or an impediment to their rapid progress.

However, even an application that is so small that it does not require formal design or independent testing still requires configuration management and change management during its maintenance phase. Often, a company will have a whole assortment of small applications that collectively constitute a significant investment in development effort.

If even one end-user's ability to do his or her work comes to depend on the availability and functionality of a small application, maintenance will be necessary some day. As time passes, the environment in which an application executes will change, some external interface will change, or the set of requirements met by the application will change. In each case, program maintenance will be required to solve problems resulting from these changes.

Application maintenance would be impossible if the application source and instructions to rebuild it have not been tightly controlled since the application first went into production use.

13.6 Interaction with Development Methodologies

Source code is only one expression of the application. The type of objects that constitute a preliminary development baseline will depend on the development methodology chosen by the project. The milestone events and the types of baselines may vary, but the principles of configuration management and change management will not.

If a project applies the traditional waterfall methodology, the baselines controlled might be:

- ☐ Functional and interface requirements
- ☐ Build-to system, subsystem, and component designs
- ☐ As-built test, integration, and delivery implementations

In the waterfall case the objects managed by configuration management and change management might be files containing:

- ☐ Various requirements specifications
- ☐ Various forms of design notation and data definitions
- ☐ Source and executable application code

If a more recent methodology, such as the Grady Booch object-oriented methodology, is used, the baselines might be:

- ☐ Conceptualization prototype
- ☐ Analysis description that models the behavior of the application
- ☐ Architectural release and descriptions of tactical policies
- ☐ Successively refined executable releases

The objects controlled by configuration management and change management in the object-oriented case might be files containing:

- ☐ Object and class diagrams, finite state machines, and documented nonbehavioral aspects of the design such as portability, reliability, security, and efficiency
- ☐ Class and object structure diagrams and an architectural release
- ☐ Source and executable application code

Configuration management and change management procedures and mechanisms must be tailored to meet the requirements of the development methodology.

13.7 *Interaction with Project Management*

Configuration management and change management provide input and assistance to project management. For example, an upgrade to the application may imply a change in the set of defined activities or milestones. The investigation performed while evaluating the impact of a given change will alert project management of a need to make changes in the development schedule or apply additional human resources to the project.

Configuration management and change management mechanisms can be used to ensure consistency across the application in many ways such as checking the presence and content of module headers, recording approval of quality assurance representatives, and ensuring that related design updates, test results, software quality metrics data and upgraded end-user documentation are submitted with related code changes.

Change management mechanisms provide for input from various members of a project including management itself. Change management provides a means by which management can assign responsibility for the implementation of changes as well as monitor their progress.

13.8 Interaction with Quality Assurance

Configuration management and change management implement the specific policies and practices espoused and adopted by a development effort. Quality assurance organizations exist to inspect and verify that the effort conforms to those policies and practices as well as any externally applied rules and regulations. Quality assurance oversight of the application development process is very much enhanced by formal configuration management and change management procedures. These procedures can be implemented in such a way as to ensure that software does not enter a baseline unless it has been approved by quality assurance representatives and is accompanied by the required supporting material, such as test plans and test results. These representations can ensure that changes to a formal baseline are always accompanied by identification of the corresponding problem report. They can also ensure that software conforms to project quality standards regarding such topics as module headers, minimum number of lines of comments, and naming conventions. Quality assurance representatives can easily inspect and approve these procedures and the resulting data.

13.9 Configuration Management History and Statistics

If configuration management and change management are implemented in connection with a database providing generalized query capability, risk, cost, quality metrics, and other data can be captured and analyzed for project management uses also. Various statistics related to project management, software quality configuration management, and change management can be derived from a well-maintained configuration management database. It is possible to extract sufficient data from such a database to project cost, staffing, software sizing, and development schedule data of similar projects under planning.

Figure 185 shows the relationships among configuration management, change management, and the application development environment.

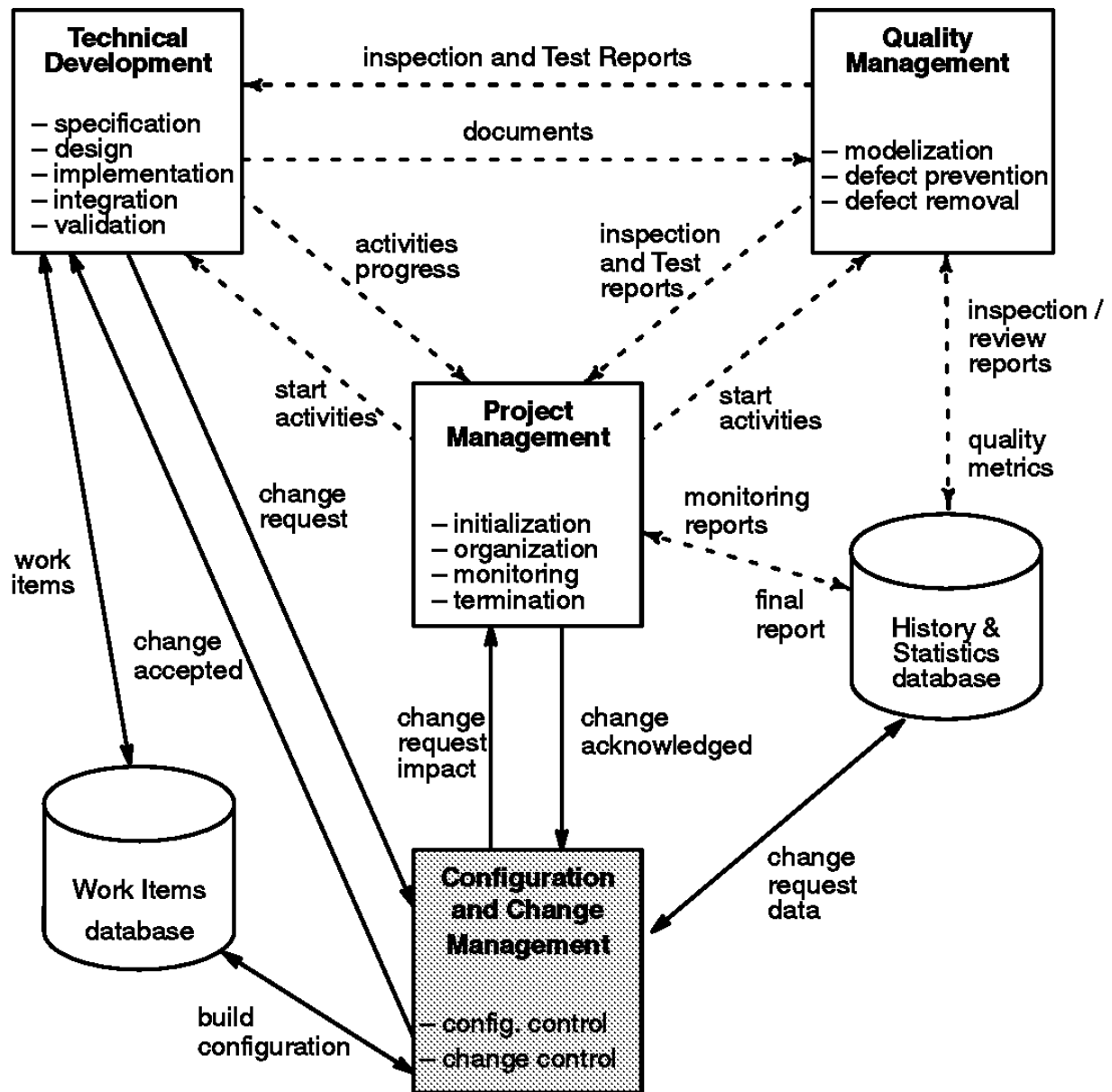


Figure 185. Development Process Relationships

Chapter 14. Implementation of ISO 9001 Using TeamConnection

14.0 Chapter 14. Implementation of ISO 9001 Using TeamConnection

This chapter provides a brief introduction to International Organization for Standardization (ISO) 9000 and describes how the IBM TeamConnection tool can be used to meet some key ISO 9001 requirements.

The software engineering industry is the fastest growing industry in the last half of this century. Throughout the industry, software development organizations are struggling with the challenges of reducing costs, increasing productivity, and improving quality. Toward these efforts, quality management of software is essential. One way to establish a quality management system is to provide guidance for software quality assurance. Such guidance is found in the ISO 9000 series of quality standards: ISO 9001, ISO 9002, ISO 9003.

ISO 9000 compliance is of key importance to organizations if they are to survive the fierce competition of the 1990s and beyond. With the introduction of ISO 9000, the software engineering industry has experienced a shift toward implementing techniques and processes aimed at developing processes that are well defined and repeatable. Adoption of these proven techniques and processes allows an organization to improve the overall process of:

- ☐ Creating world class software
- ☐ Maintaining a dynamic, responsive, and innovative environment
- ☐ Attaining a high return on investment through the pursuit of total customer satisfaction

Software tools can assist in improving an organization's management system and meeting ISO 9000 compliance. TeamConnection is an effective tool that simplifies the organization and management of diverse tasks involved in software development so that you can improve your entire product development process and TeamConnection can be used to comply with some key elements of ISO 9001.

Subtopics

14.1 ISO 9000

14.2 TeamConnection and ISO 9001

14.3 Conclusion

14.4 Brief Description of ISO 9000-3

14.5 References

14.1 ISO 9000

To facilitate the standardization of the many aspects of quality, the ISO has developed a set of international standards for quality systems known as the ISO 9000 Series of Quality Standards. These standards apply to all organizations producing a product or service and are being accepted worldwide. An indication of their acceptance and significance in the software engineering industry, in particular, is reflected in Hübner's words, "To obtain an ISO 9000 certification has become a business necessity in Europe" [1]. It is only a matter of time before ISO 9000 certification becomes a business necessity in North America and the Orient.

Three key ISO 9000 standards are:

- ☐ ISO 9001, Quality systems - Model for quality assurance in design/development production, installation, and servicing
- ☐ ISO 9002, Quality systems - Model for quality assurance in production and installation
- ☐ ISO 9003, Quality systems - Model for quality assurance in final inspection and test

ISO 9001 provides the most comprehensive requirements for a software quality system where a contractual agreement between two parties must demonstrate the supplier's ability to design and supply a product or service.

Recognizing the peculiarities of the software industry, ISO 9000-3, a guideline for the application of ISO 9001 to the development, distribution and maintenance of software, has been released. Configuration management is a key element in this ISO 9000-3 guideline for software.

14.2 TeamConnection and ISO 9001

Configuration management provides a mechanism for identifying, controlling, and tracking the versions of each software item. In many cases, multiple versions of software items are in use and must be maintained and controlled. Configuration management itself is not an element of the ISO 9001 standard (only an element of ISO 9000-3 guidelines); however, the following elements of ISO 9001 depend on configuration management:

- ☐ Document Control
- ☐ Design Control
- ☐ Product Identification and Traceability
- ☐ Inspection and Test Status
- ☐ Control of Nonconforming Product
- ☐ Internal Quality Audits

Only certain aspects of Document Control, Design Control, Control of Nonconforming Product, and Internal Quality Audits are addressed by configuration management. Product Identification and Traceability and Inspection and Test Status are fully addressed by it.

The remaining sections describe these ISO 9001 elements and highlight how TeamConnection can support them.

Subtopics

- 14.2.1 Document Control
- 14.2.2 Version Control in ISO 9001
- 14.2.3 Internal Quality Audits

14.2.1 Document Control

Document Control covers:

- ☐ The determination of those documents that should be subject to document control procedures
- ☐ The approval and issuance of document control procedures
- ☐ The change procedures including withdrawal and, as appropriate, release [2]

The aspects of Document Control that can be successfully addressed by TeamConnection are:

- ☐ Documents must be accessible to a group of people with predetermined interest and authority.
- ☐ The changes to the content of a document under document control have to be reviewed by a prespecified group of reviewers, and the final version of the document has to be approved by them.
- ☐ All users of the document have to be notified of changes to it.
- ☐ Once the new document is finalized and becomes the most recent working document, provisions must be taken to prevent users from using a back-level version of the same document.

Files and documents pertaining to a particular project reside in one or more TeamConnection components (where each TeamConnection component is dedicated to a specific department and/or all documents related to a particular project).

An access list and a notification list are associated with each TeamConnection component. The type of access each user has to the documents stored in TeamConnection depends on the user's role in the development team. The type of notification each user has depends on the interest in or need to be informed of changes to documents in the TeamConnection environment. Access authority and notification subscription are assigned to a user by the component owner or by someone who has the authority to grant other users access and notification to a specific component. When a document is updated, the owner of the managing component where the document resides and all users who have subscribed to being informed of document updates receive mail notifying them of the update.

When the TeamConnection approval process is activated, approval must be given for proposed changes before work can begin on the implementation of a change. Approvers specified for each release must review the information recorded in the defect or feature and evaluate the proposed changes to the release in relation to other project considerations. A TeamConnection approval record is created for each approver. Each approver indicates his or her evaluation of the changes and can optionally append comments to the defect or feature to explain the rationale for his or her decision. File changes for that defect or feature in that release cannot be checked in to the TeamConnection server until all approvers accept the proposed changes. [3]

14.2.2 Version Control in ISO 9001

Certain aspects of Design Control, Product Identification and Traceability, Inspection and Test Status, and Control of Nonconforming Product deal with the issue of *version control*. Activities are versioning documents and source modules that compose a product; identifying the various versions of the documents and source modules and the reasons why changes were made from one version to the next; inspecting and testing the content of each version and recording the status of this outcome; and controlling nonconforming products and identifying the version of documents and source modules that reflect the nonconformance.

Version control, by definition, is the storage of multiple versions of a single file along with information about each version [3]. TeamConnection provides for version control and enhances this basic function with an extra layer of traceability so that each version is cross-referenced to a reported defect or a suggested enhancement.

The following sections highlight the specific sections of each of the ISO 9001 elements that emphasize the need for version control mechanisms in an organization.

Subtopics

- 14.2.2.1 Design Control
- 14.2.2.2 Product Identification and Traceability
- 14.2.2.3 Inspection and Test Status
- 14.2.2.4 Control of Nonconforming Product

14.2.2.1 Design Control

The ISO 9001 element of Design Control states that "the supplier shall establish and maintain procedures to control and verify the design of the product in order to ensure that the specified requirements are met." One of the items that define this element of Design Control is the Control of the Design Changes, which is defined as:

```
+-----+
|
| "The supplier shall establish and maintain procedures for the
|   identification, documentation and appropriate review and approval
|   of all changes and modifications." [4]
|
+-----+
```

TeamConnection can be used to control and verify the design of a product in a number of ways. First, the design specification documents themselves can be stored in TeamConnection. Modifications to the content of the design specifications can be controlled from both an access and an update perspective. Development teams can make use of the problem-tracking feature of TeamConnection to track and control the changes to design specifications and to ensure that all changes have been well documented, justified, and reviewed for appropriateness and applicability. TeamConnection's design, size, and review process for reported defects and suggested features provides for the identification, documentation, and appropriate review of all changes and modifications. TeamConnection's tracking process allows development teams to cross-reference changes to design documents or source modules to the reported defects and features. It also provides an additional layer of control to those teams seeking an approval checkpoint before versioning the documents and a review of the changes after making the modifications but before committing them.

When source modules for a product are managed by TeamConnection, development teams can ensure that the changes made to the product are consistent with its design by using the integrated problem-tracking and change control feature of TeamConnection. Attributes of reported defects and features can be used to cross-reference design documents with the proposed source module changes to the product. TeamConnection's ability to track required changes in all project deliverables ensures that appropriate updates are made to design documents, source modules, test cases, and end-user documentation for each reported defect and suggested enhancement of the product.

When the time comes to release a product, development teams can benefit from TeamConnection's ability to maintain multiple versions or variants of a product. Enhancements for the next release can be incorporated into design documents and source modules without disrupting the integrity of the products that have been distributed.

14.2.2.2 Product Identification and Traceability

The ISO 9001 element applicable to version control for Product Identification and Traceability states:

```
+-----+
| "Where appropriate the supplier shall establish and maintain |
| procedures for identifying the product from applicable drawings, |
| specifications or other documents, during all stages of production, |
| delivery and installation. Where and to the extent that, |
| traceability is a specified requirement, individual product or |
| batches shall have a unique identification. This identification |
| shall be recorded." [4] |
+-----+
```

When TeamConnection is used as the configuration management and version control tool for software development activities, several levels of identification and traceability are available.

Each version of a document or source module is identified by a version number. The combination of this version number as well as the document or source module name (TeamConnection part name) and the product name with which the document or source module is associated (TeamConnection release name) uniquely identify a part in TeamConnection.

As previously discussed, changes to documents and source modules can be cross-referenced to TeamConnection defects and features. Users can then query the history of changes and identify the content of each version of a document or source module and the reason why it has changed over time. Alternatively, users can query the details of the defects and features to determine the document or source modules that were changed as a result of fixing a reported problem or implementing a suggested feature.

TeamConnection records the time and date of the changes to documents or source modules as well as the user who makes each change. This information provides additional traceability and can be queried at any time.

14.2.2.3 Inspection and Test Status

The ISO 9001 element applicable to version control for Inspection and Test Status states:

```
+-----+
|
| "...The identification of inspection and test status shall be
| maintained, as necessary, throughout production and installation
| of the product to ensure that only product that has passed the
| required inspections and tests is dispatched, used or installed.
| Records shall identify the inspection authority responsible for
| the release of conforming product."[4]
|
+-----+
```

The TeamConnection problem-tracking mechanism allows development teams to establish two types of testing procedures. When development teams use the tracking mechanism, they can define test environments and testers for each release of a product. As defects are fixed and features are implemented, TeamConnection activates test records for each of the test environments and testers, indicating when the changes made to documents and source modules have been committed in the product. Testers are notified when their test records are ready to be marked. By marking a test record with an accept, reject, or abstain status, a tester relays information to the development team as to the status of the change. When test records are rejected, additional defects or features can be opened to track the nonconformances, and attributes in both the original and the new defects or features can be used to cross-reference problems of a similar nature.

Another type of testing occurs at the end of the TeamConnection defect or feature life cycle. Once applicable documents or source modules have been changed and committed into the various products, the originator of the defect or feature has the opportunity to verify that the resolution of the problem or the implementation of the suggestion has been accomplished to his or her satisfaction. Originators record their satisfaction of the outcome on verification records.

The status of test and verification records, the owner of test and verification records, and the time stamp of when each record was last updated are maintained in TeamConnection. The reporting mechanism allows users to query the status of these records at any time.

14.2.2.4 Control of Nonconforming Product

The ISO 9001 element applicable to version control for the Control of Nonconforming Product states:

"....Control shall provide for identification, documentation, evaluation, segregation (when practical), disposition of nonconforming product and for notification to the functions concerned." [4]

Nonconformances with respect to products managed by TeamConnection are identified by opening a TeamConnection defect or feature. Once the nonconformances are identified, the development teams can evaluate the validity of the nonconformance and return the defect or feature as invalid, as a documented deviation, or as a nonconformance that has already been addressed by another defect or feature report. Alternatively, the development team can decide to accept responsibility for the nonconformance and schedule its resolution in the appropriate product releases. In either case, all users who have subscribed to defect and feature reports and state changes will be kept informed.

When nonconformances are received for products that have been released to customers, development teams can resolve the nonconformance and reissue a product update. Development teams can make use of the attributes associated with product levels to describe the status of the package. For instance, a product level may be shipped and then subsequently replaced with a newer version that includes fixes for nonconformances.

14.2.3 Internal Quality Audits

```
+-----+
| ".....The audits and follow-up actions shall be carried out in |
| accordance with documented procedures.  The results of the audits |
| shall be documented and brought to the attention of the personnel |
| having responsibility in the area audited.  The management personnel|
| responsible for the area shall take timely corrective action      |
| on the deficiencies found by the audit." [4]                      |
+-----+
```

A company could use internal audits to check the level of compliance of the various departments. Internal audits could be conducted and nonconformances issued and monitored until a satisfactory corrective action plan is put in place and successfully implemented.

In a specific area, a component dedicated to ISO 9001 can be created and an owner assigned to it. Traditionally, the owner of this component is the ISO focal point for the area. Each department in the area has its own component where all the documents and processes are stored.

The internal audit group could open defects (Nonconformances) against the area component. The component owner could then route the nonconformances to the corresponding department component. The departments would be responsible for creating their own corrective action plan. Each corrective action is appended to the nonconformance in TeamConnection, and all the interested parties are notified that remarks were appended to the specific nonconformance.

The remarks added to each of the nonconformances that described the corrective action are retrieved through TeamConnection and then forwarded to the internal audit group for review. This group creates the actual corrective action plan for each department and hence for the area as a whole.

14.3 Conclusion

To make software quality a reality and to comply with the ISO 9001 standard, TeamConnection can successfully be used for:

- ☐ Design Control
- ☐ Document Control
- ☐ Product Identification and Traceability
- ☐ Inspection and Test Status
- ☐ Control of Nonconforming Product
- ☐ Internal Quality Audits

14.4 Brief Description of ISO 9000-3

This section describes the ISO 9000-3 elements that pertain to configuration management, design control, and document control.

Subtopics

14.4.1 Configuration Management

14.4.2 Design Control

14.4.3 Document Control

14.4.1 Configuration Management

Configuration Management should:

- ☐ Uniquely identify the versions of each software item
- ☐ Identify the versions of each software item that together constitute a specific version of a complete product
- ☐ Identify the build status of software products in development or the status of those software products delivered and installed
- ☐ Control simultaneous updating of a given software item by more than one person
- ☐ Provide coordination for the updating of multiple products in one or more locations as required
- ☐ Identify and track all actions and changes resulting from a change request, from initiation through to release

Subtopics

14.4.1.1 Configuration Identification and Traceability

14.4.1.2 Change Control

14.4.1.3 Configuration Status Report

14.4.1.1 Configuration Identification and Traceability

To establish and maintain procedures for identifying software items during all phases, starting from specification through development, replication and delivery, each individual software item should have a unique identification.

There should be provisions to uniquely identify the following items for each version of the software:

- ☐ The functional and technical specifications
- ☐ All development tools that affect the functional and technical specifications
- ☐ All interfaces to other software and/or hardware items
- ☐ All documents and computer files related to the software item

The identification of a software item should be handled in such a way that the relationship between the item and the contract requirements can be demonstrated.

For released products, there should be procedures to facilitate traceability of the software item or product.

14.4.1.2 Change Control

Procedures should be in place to identify, review, and authorize any changes to the software items under the control of configuration management. All changes to software items should be carried out in accordance with these procedures.

Before a change is accepted, its validity should be confirmed, and the effects on other items should be identified and examined.

Methods to notify those concerned of the changes and to show the traceability between changes and modified parts of software items should be provided.

14.4.1.3 Configuration Status Report

The supplier should establish and maintain procedures to record, manage, and report on the status of software items, change requests, and the implementation of approved changes.

14.4.2 Design Control

The supplier should establish and maintain procedures to control and verify the design of the product in order to ensure that the specified requirements are met.

14.4.3 Document Control

Procedures should be established and maintained to control all documents that relate to the contents of this part of ISO 9000. They cover:

1. The determination of those documents that should be subject to the document control procedures.
2. The approval and issuance of document control procedures.

All documents should be reviewed and approved by authorized personnel prior to issue. Procedures should exist to ensure that the pertinent issues of appropriate documents are available at appropriate locations where operations essential to the effective functioning of the quality system are performed. Obsolete documents should be promptly removed from appropriate points of issue or use.

Where use is made of computer files, special attention should be paid to appropriate approval, access, distribution, and archiving procedures.

3. The change procedures including withdrawal and, as appropriate, release

14.5 References

1. Hübner, Achim, *ISO 9000 Implementation in Germany*, **LOGON**, Volume 4, Number 4, September 1992 (IBM Internal Use)
2. *International Standard: ISO 9000-3*, Reference Number ISO 9000-3:1991(E)
3. *IBM CMVC Concepts*, SC09-1633-00, IBM Corporation, 1993.
4. *International Standard: ISO 9001*, Reference Number ISO 9001:1987(E).

A.0 Appendix A. Component Hierarchy Creation Aid

Use Table 9 as an aid and a template when planning and preparing for your component hierarchies. The related command line parameters are given in the table, so that you can easily create a REXX or OS/2 command file to help you with the creation of your component hierarchies.

The following TeamConnection commands used to create and maintain components and define component hierarchies in a family:

- This command creates components with the specified names:

```
teamc component -create Name ... -parent Name -process Name
                -family Name
                [-owner Name] [-description Text]
                [-become Name] [-verbose]
```

Component names must be unique within a family.

- This command deletes the specified components:

```
teamc component -delete Name ... -family Name
                [-become Name] [-verbose]
```

The *root* component cannot be deleted.

- This comand attaches components to an existing component:

```
teamc component -link Name ... -parent Name -family Name
                [-become Name] [-verbose]
```

The components you list with the **-link** flag become child components of the component you specify with the **-parent** flag.

- This command re-creates components as child components of the parent component:

```
teamc component -recreate Name ... -parent Name -family Name
                [-become Name] [-verbose]
```

Use the **-parent** flag to specify the parent.

- This command detaches components from a parent component:

```
teamc component -unlink Name ... -parent Name -family Name
                [-become Name] [-verbose]
```

The components being unlinked must still be linked to at least one parent component.

In all of the above commands, "... " indicates that additional names are allowed.

[illegible]

[illegible]

Notes: TeamConnection parameters in parentheses. See the *IBM TeamConnection for OS/2 Commands Reference* SC34-4501, for an explanation of the TeamConnection command line parameters.

B.0 Appendix B. Build Tree Aid

Use Table 10 as an aid and a template when planning and preparing for build tree creations. The related command line parameters are given in the table, so that you can easily create a REXX or OS/2 command file to help you with build tree creation.

The following TeamConnection commands are used to manage the *build trees*:

- This command shows the immediate children (of the part) in the build tree:

```
teamc part -childInfoView Name ... -release Name -family Name
[-long] [-workarea Name*** | -version Name]
[-type Name] [-become Name] [-verbose]
```

- This command connects a part to a parent in the build tree:

```
teamc part -connect Name ... -parent Name
{-input* | -output | -dependent}
-workarea Name -release Name -family Name
[-type Name] [-parenttype Name]
[-become Name] [-verbose]
```

- This command creates parts with the specified names:

```
teamc part -create Name ... -component Name
          -workarea Name -release Name -family Name
          [-builder Name] [-fmod Octal_number]
          [-relative Name | -top Name]
          [-binary | -text* | -none]
          [-parameters Parameters]
          [-parent Name {-input* | -output | -dependent}]
          [-parser Name]
          [-stdin | -from fileSpec | -empty ]**
          [-parenttype Name*]
          [-remarks Text]
          [-temporary* | +temporary]
          [-become Name] [-verbose]
```

A part must have a unique path name and type within a release.

- This command disconnects the part from its parent in the build tree:

```
teamc part -disconnect Name ... -parent Name
           -workarea Name -release Name -family Name
           [-type Name] [-parenttype Name]
           [-become Name] [-verbose]
```

In the above commands:

```
* - default if not specified
  -type and -parenttype will default to file if not specified
** - Only valid with file type of text or binary.
*** - Required if part has not been integrated
... - Additional names are allowed
```

Note: The `part -create` command is the only command that uses the `-component` attribute.

So if in **family** *fam2008*, **release** *AFT-MVS*, and **work area** *aftmvs* you wanted to connect the *AFTMVS.obj*, *AFTSNA.obj*, *AFTFILIO.obj*, *AFTARG.obj*, *APPCMVS.load*, and *APPCATTN.load* **parts** to the *AFTMVS.load* **parent**, you would issue the following command:

```
teamc part
-connect AFTMVS.obj AFTSNA.obj AFTFILIO.obj AFTARG.obj APPCMVS.load
        APPCATTN.load
-parent AFTMVS.load
-family fam2008 -release AFT-MVS -workarea aftmvs
```

Table 10. Build Tree Aid				
Family (-family)		Release (-release)		Work Area (-workarea)

Parent Name (-parent)	Type (-parenttype)	Part Name (-connect)	Type (-type)	Component Name

This image shows a full page of graph paper. It features a grid of dashed lines forming squares. There are 20 horizontal rows and 5 vertical columns. In the top right corner, there is a small, partially visible label that reads "(- compor".

Notes: TeamConnection parameters in parentheses. See the *IBM TeamConnection for OS/2 Commands Reference*, SC34-4501, for explanation of the TeamConnection command line parameters.

C.0 Appendix C. Useful REXX Programs

This appendix contains the following REXX programs, which you will find useful when working with TeamConnection:

- ☐ rct_fam.cmd
- ☐ strt2008.cmd
- ☐ tcchkin.cmd
- ☐ tcchkout.cmd
- ☐ tcparts.cmd
- ☐ tcextract.cmd

```
+--- Caution -----+
|
| The REXX programs in this appendix are supplied on an as-is basis.
|
+-----+
```

In the sections that follow we list the programs and explain their functions.

Subtopics

- C.1 A Program to Re-create the Family
- C.2 A Program to Start the Family
- C.3 A Program to Check In Multiple Parts
- C.4 A Program to Check Out Multiple Parts
- C.5 A Program to Create Multiple Parts
- C.6 A Program to Extract Multiple Parts

C.1 A Program to Re-create the Family

This REXX program is used to re-create a family if for some reason something has gone wrong and you have to re-create most of your work. The program reads the *teamc.log* (or any TeamConnection log file) and tries to re-create the family according to the information in the log file.

Note: This program cannot handle modification-type actions, such as; check in, check out, or edit.

Figure 186 shows the *rct_fam.cmd* REXX program.

```

-----
/*****
/* Last update: 06/26/96
/* Author      : Leif Trulsson, IBM ITSO San Jose
/* Enhanced by: Benno Muilwijk, IBM Uithoorn
/* Use RCT_FAM ? for help
*****/

trace o
'@echo off'
'setlocal'                                /* save environment */

parse upper source . . cmdname
cmdname = substr(cmdname, lastpos('\', cmdname) + 1)
cmdname = left(cmdname, lastpos('.', cmdname) - 1)
logname = cmdname'.LOG'

parse arg input_string

if pos('?', input_string) > 0 then signal HELP /* Help & QUIT */

call initialize

i = 1
do while lines(infile)                    /* read in the log file */
  log.i = linein(infile)
  if word(log.i, 1) = '===' then
    log.i = subword(log.i, 2)
  else
    log.i = strip(log.i)
  select
    when word(log.i, 1) <> 'teamc' then nop
    when word(log.i, 2) = 'Report' then nop
    when word(log.i, 2) = 'Part' &,
      pos('-create', log.i) = 0 &,
      pos('-delete', log.i) = 0 &,
      pos('-destroy', log.i) = 0 &,
      pos('-recreate', log.i) = 0 &,
      pos('-connect', log.i) = 0 &,
      pos('-disconnect', log.i) = 0 &,
      pos('-link', log.i) = 0 &,
      pos('-modify', log.i) = 0 &,
      pos('-rename', log.i) = 0 then nop
    when pos('-raw', log.i) > 0 then nop
    when pos('-configInfo', log.i) > 0 then nop
    when pos('-extract', log.i) > 0 then nop
    when pos('-view', log.i) > 0 then nop
  otherwise
    i = i + 1
  end /* select */
end /* do */
rc = stream(infile, 'C', 'CLOSE')          /* close the log file */
log.0 = i - 1

/* setup the new log */
logbkup = left(logfile, length(logfile) - 1) || '2'
'del' logbkup '1>NUL 2>NUL'
'ren' logfile logbkup '1>NUL 2>NUL'

do i = 1 to log.0
  call runCmd log.i
end /* do */

'endlocal'                                /* restore environment */
exit 0

/*****
*****/
/* runCmd
*****/
-----

```

```

/* Executes a command and logs the results in a the */
/* 'rc_fam.log' file */
/* */
/* It will also beep if there is an unexpected result. */
/***** */
runCmd:

    parse arg cmd

    /* log and run the command */
    say '===' cmd
    call lineout logfile, '===' cmd

    if test = 'N' then do
        if pos('>', cmd) > 0 then trace r
        cmd '1>buf.out 2> buf.err'
        saverc = rc

        /* check if result is expected */
        if saverc <> 0 then do
            BEEP(440,200)

            /* type results to the screen */
            say '--- rc='saverc
            'type buf.out'
            'type buf.err'
        end /* do */

        /* type results to the logfile */
        call lineout logfile, '--- rc='saverc
        call lineout logfile
        'type buf.out >>' logfile
        'type buf.err >>' logfile

        trace o
    end /* Do */

    call lineout logfile, ''
    call lineout logfile

return

/***** */
/* Setup up the test environment */
/***** */
initialize:

/* say 'input_string =' input_string */

family = ''
user = ''
infile = ''
outpath = ''
test = 'N'

do while input_string <> ''
    parse var input_string next input_string
    parse upper var next next .

    select
        when next = '-F' then
            call family
        when next = '-U' then
            call user
        when next = '-I' | next = '-P' then /* -P for compatibility */
            call infile
        when next = '-O' then
            call outpath
        when next = '-D' then
            call test
        otherwise
            signal HELP
    end /* select */

end /* do */

if family = '' then
    family = getenv('TC_FAMILY')
if user = '' then
    user = getenv('TC_USER')
if infile = '' then
    infile = getenv('TC_LOGFILE')
if outpath = '' then

```

```

    outpath = getenv('TC_LOGPATH')
If outpath <> '' then
    If Right(outpath,1) = '\' | Right(outpath,1) = ':' then
        nop
    else
        outpath = outpath\'
logfile = outpath || logname

say 'family      =' family
say 'user        =' user
say 'input log   =' infile
say 'output log  =' logfile
say 'test run    =' test
say
if family = '' | infile = '' then do
    say 'One or more of the parameters was not specified and the default'
    say 'could not be determined from the corresponding environment variable.'
    say
    say 'Press Enter to view the help information.'
    pull answer
    signal HELP
end

say 'Correct? Y/N'

pull answer

if answer <> 'Y' then
    signal ENDIT

return

/*****/
/* check that variables are set */
/*****/

family:
    parse var input_string next input_string
    if next = '' | left(next,1) = '-' then
        signal HELP
    family = next
    'SET TC_FAMILY='family
return

user:
    parse var input_string next input_string
    if next = '' | left(next,1) = '-' then
        signal HELP
    user = next
    'SET TC_USER='user
return

infile:
    parse var input_string next input_string
    if next = '' | left(next,1) = '-' then
        signal HELP
    infile = next
return

outpath:
    parse var input_string next input_string
    if next = '' | left(next,1) = '-' then
        signal HELP
    outpath = next
return

test:
    test = 'Y'
return

getenv:
    arg envvar
return value(envvar,, 'OS2ENVIRONMENT')

/*****/
FAILED:
say 'Error in writing line:' i
signal ENDIT

/*****/
HELP:
call RxFuncAdd 'SysCls', 'RexxUtil', 'SysCls'
call SysCls

```

```

say
say '*'-----*
say '*      TEAMC OS/2 Family recreate      *'
say '* This Rexx program recreates a family by using the *.log files. *'
say '* Author: Leif Trulsson, IBM ITSO San Jose - The Redbook People *'
say '*'-----*
say
say 'SYNTAX:' cmdname '<-f family> <-u user> <-i infile> <-o outpath> <-d>'
say 'Where : '
say '  -f family   = The name of the family.'
say '              Default is the value of environment variable TC_FAMILY.'
say '  -u user      = The name of the superuser.'
say '              Default is the value of environment variable TC_SER.'
say '  -i infile    = The name of the input log file created by either'
say '                TeamConnection or a previous run of ' cmdname '.'
say '              Default is the value of environment variable TC_OGFILE.'
say '  -o outpath   = The name of the path where the output log file'
say '                ('logname') will be stored.'
say '              Default is the value of environment variable TC_OGPATH.'
say '  -d           = Debug test mode, does not run the commands.'
say
say
say /*****
say /* Exit program */
say /*****/
Endit:
say /* call DropLs30outFuncs */
say /* call SysDropFuncs      */
say 'endlocal'                  /* restore environment */
say Exit 0

```

Figure 186. The rct_fam.cmd Program

C.2 A Program to Start the Family

This REXX program is used to automatically start the family. The program also starts any build agents and build processors that have to be started on the TeamConnection server machine. Figure 187 shows the *strt2008.cmd* REXX program.

```

/*-----*\
          Start fam2008
\*-----*/

family = "fam2008"
db_drive = "G:"
db_path = "\fam2008"

" " db_drive
"CD" db_path

"SET TC_FAMILY="family
"SET TC_DBPATH="db_drive||db_path"\family
"SET TC_CACHESIZELIMIT="10*1024*1024
"SET TC_CACHEPRUNEMETHOD=DATE"

'START "TEAMC server for:' family '" /MIN',
      'TEAMCD.EXE -p bldprcs.lst',      /* build processors */
      '-a bldagnts.lst',                /* build agents      */
      '-n mailexit.cmd',                /* Notify Mail Exit  */
      family '3'                        /* Number of daemons */

EXIT 0

```

Figure 187. The *strt2008.cmd* Program

The *strt2008.cmd* program uses two files to start the build agents and the build processors:

- ☐ **bldagnts.lst**
- ☐ **bldprcs.lst**

Figure 188 and Figure 189, respectively, show the two files.

```

#-----
# Build Agents for fam2008
#
# -s socket port
# -e environment
# -p build pool
# < -k local codepage >
#
#-----
-s bld2008 -e os2 -p pool1

```

Figure 188. The *bldagnts.lst* File

```

#-----
# Build processors for fam2008
#
# -s build socket
# < -c cache location >
# < -k local codepage >
# < -n > (erase cache directory before starting build processor)
#
#-----
-s bld2008 -c g:\cache\fam2008\c1 -n

```

+-----+

Figure 189. The bldprcs.lst File

C.3 A Program to Check In Multiple Parts

This REXX program can check in multiple parts. Figure 190 shows the *tcchkin.cmd* REXX program.

```

+-----+
/*****
/*      TEAMC OS/2 Part Check-in program      */
/*****
/* This REXX program checks in PART(s) to a given      */
/* workarea, release and family from a specified      */
/* directory                                          */
/*                                          */
/* It is broken up into 3 sections.                  */
/* 1. TEAMC initialization                          */
/* 2. TEAMC check in PARTS                          */
/* 3. TEAMC Report on PARTS                          */
/*                                          */
/*****
/* Last update: 06/26/96                          */
/* Author      : Leif Trulsson, IBM ITSO San Jose    */
/*****
call RxFuncAdd 'SysCls' , 'RexxUtil', 'SysCls'
call RxFuncAdd 'SysTextScreenSize' , 'RexxUtil', 'SysTextScreenSize'
call RxFuncAdd 'SysCurPos' , 'RexxUtil', 'SysCurPos'
call RxFuncAdd 'SysFileTree' , 'RexxUtil' , 'SysFileTree'

env='OS2ENVIRONMENT'

parse arg input_string

/* Set up common variables */

family = ''
name = ''
become = ''
release = ''
workarea = ''
sourcedir = ''
listfile = ''
test = 'N'

call initialize

call PARTS

exit

/*****
/*      Functions      */
/*****

PARTS :

/* Check if we have a 'listfile' or not as one of the parameters */
if listfile = ' ' then do /* If not... */
    rc = SysFileTree(name , 'filelist' , 'FO')
    do i = 1 to filelist.0
        fullfile = filelist.i
        name = filespec('NAME',fullfile)
        say 'Part name =' name

        command = 'Part -checkin '||name||' -family '||family||' -workarea '||workarea||' -release ' ,
                    ||release||become||sourcedir||' -v||rbose'

        call runCmd command

    end
end /* Do */
else do /* If we have, we come here */
    i = 0
    do while lines(listfile) /* read in the listfile */
        i = i + 1
        log.i = linein(listfile)
        fullfile = log.i
        name = filespec('NAME',fullfile)

        sourcedrive= filespec('DRIVE',fullfile)

        if sourcedrive <> ' ' then do
            sourcedir= filespec('PATH',fullfile)
            sourcedir = ' -relative '||sourcedrive||sourcedir

```



```

end /* do */

command = 'Part -checkin '||name||' -family '||family||' -workarea '||workarea||' -release '||release||become||sourcedir||' -verbose'

call runCmd command
end /* do */
Myrc=stream(listfile,'c','close') /* close the listfile */
end /* Do */

report = 'Report -view partfullview -family '||family||' -where "release = '||release||', -verbose'

call runCmd report

return

/*****
/* Process input parameters and initialize variables */
*****/

initialize:

say 'input_string =' input_string
do forever while input_string <> ""
  parse var input_string next input_string
  parse upper var next next dummy

  select
    when next = '-B' then
      call BECOME
    when next = '-F' then
      call FAMILY
    when next = '-L' then
      call LISTFILE
    when next = '-N' then
      call NAME
    when next = '-R' then
      call RELEASE
    when next = '-S' then
      call SOURCE
    when next = '-D' then
      call TEST
    when next = '-W' then
      call WORKAREA
    otherwise
      call HELP

  end /* select */
end /* do */

/* Check input parameters for values if none set default */

if family = '' then
  family = value('TC_FAMILY','','OS2ENVIRONMENT')
if name = '' then
  name = '.*'
if become = '' then
  become = value('TC_BECOME','','OS2ENVIRONMENT')
if release = '' then
  release = value('TC_RELEASE','','OS2ENVIRONMENT')
if sourcedir = '' then
  sourcedir = value('TC_SOURCE','','OS2ENVIRONMENT')
if workarea = '' then
  workarea = value('TC_WORKAREA','','OS2ENVIRONMENT')

/* Display what we got and ask user for Y or N */

say 'name      =' name
say 'family    =' family
say 'release   =' release
say 'workarea  =' workarea
say 'become    =' become
say 'sourcedir =' sourcedir
say 'listfile  =' listfile
say 'test run  =' test
say ' '
say 'Correct ? Y/N '

parse upper pull answer

if answer = 'Y' then
  nop
else

```

```

exit 0

/* If some of the mandatory parameters are missing, call help */

if family = '' | name = '' | release = '' | workarea = '' then
    call help

if become <> ' ' then
    become = ' -become '||become

if sourcedir <> ' ' then
    sourcedir = ' -relative '||sourcedir

/* setup the log */
logfile = 'tcchkin.log'
'@copy tcchkin.log tcchkin.lo2 1>NUL 2>NUL'
'@del tcchkin.log 1>NUL 2>NUL'

return

/*****
/* check that variables are set */
*****/

FAMILY:
    parse var input_string next input_string
    if next <> ' ' then
        family = next
    else
        call help
return

NAME:
    parse var input_string next input_string
    if next <> ' ' then
        name = next
    else
        call help
return

BECOME:
    parse var input_string next input_string
    if next <> ' ' then
        become = next
    else
        call help
return

RELEASE:
    parse var input_string next input_string
    if next <> ' ' then
        release = next
    else
        call help
return

WORKAREA:
    parse var input_string next input_string
    if next <> ' ' then
        workarea = next
    else
        call help
return

SOURCE:
    parse var input_string next input_string
    if next <> ' ' then
        sourcedir = next
    else
        call help
return

LISTFILE:
    parse var input_string next input_string
    if next <> ' ' then
        listfile = next
    else
        call help
return

TEST:
    test = 'Y'
return

```

```

/*****
/*          runCmd          */
/*****
/* Executes a command and logs the results to the      */
/* 'tcchkin.log' file                                   */
/*                                                     */
/* It will also beep if there is an unexpected result. */
/*****
runCmd:

    parse arg  cmd

    cmd = 'teamc '||cmd
    call lineout logfile, ' '
    call lineout logfile

    /* log and run the command */
    "@echo === " cmd
    "@echo === " cmd ">>" logfile
    if test = 'N' then do
        "@cmd "l>buf.out 2> buf.err"
        saverc = rc

        /* check if result is expected */
        if saverc <> 0 then do
            BEEP(440,200)

            /* type results to the screen */
            "@echo --- rc=" saverc
            "@type  buf.out"
            "@type  buf.err"
        end /* do */

        /* type results to the logfile */
        "@echo --- rc=" saverc ">>" logfile
        "@type buf.out >>" logfile
        "@type buf.err >>" logfile

    end /* Do */

    call lineout logfile, " "
    call lineout logfile

return

help:

say '-----' *'
say '*          TEAMC OS/2 Server Parts check-in      *'
say '* This Rexx program checks in the PARTS to your family. *'
say '* Author: Leif Trulsson, IBM ITSO San Jose - The Redbook People *'
say '-----' *'
say' SYNTAX: TCCHKIN -n name -f family -r release -w workarea '
say'                  <-s sourcedir -b become -l listfile -d>
say' Where : '
say'   -n name      = Specifies the parts to be checked out. Can be
say'                  specified in a couple of formats:
say'                  Unique name - check out part corresponding to given
say'                  name'
say'                  Generic name - check in part(s) corresponding to either
say'                  to files in path given or names specified in the listfile.'
say'                  If listfile is specified, only needs
say'                  to be specified.
say'   -f family    = Name of the family where the parts are stored.'
say'   -r release    = The name of the release.'
say'   -w workarea   = The name of the work area.'
say'   -s sourcedir  = The name of the source directory.'
say'   -b become     = The TC_BECOME user ID.'
say'   -l listfile   = File containing list of parts to be checked in.'
say'   -d           = Debug test mode, does not run the command.'
say' PRESS ENTER FOR MORE...'
parse pull dummy
say' Results of the program are written to the file TCCHKIN.LOG'
say' The program also creates a partfullview Report based on family and
say' release name.'
exit 0

```



Figure 190. The tcchkin.cmd Program

C.4 A Program to Check Out Multiple Parts

This REXX program can *check out* multiple parts. Figure 191 shows the *tcchkout.cmd* REXX program.

```

/******
/*      TEAMC OS/2 Part Check-out program      */
/******
/* This REXX program checks out PART(s) from a given */
/* workarea, release and family to a specified directory*/
/*
/* It is broken up into 3 sections.      */
/* 1. TEAMC initialization      */
/* 2. TEAMC check out PARTS      */
/* 3. TEAMC Report on PARTS      */
/*
/******
/* Last update: 06/26/96      */
/* Author      : Leif Trulsson, IBM ITSO San Jose      */
/******
call RxFuncAdd 'SysCls' , 'RexxUtil', 'SysCls'
call RxFuncAdd 'SysTextScreenSize' , 'RexxUtil', 'SysTextScreenSize'
call RxFuncAdd 'SysCurPos' , 'RexxUtil', 'SysCurPos'
call RxFuncAdd 'SysFileTree' , 'RexxUtil' , 'SysFileTree'

env='OS2ENVIRONMENT'

parse arg input_string

/* Set up common variables */

family = ''
name = ''
become = ''
release = ''
workarea = ''
targetdir = ''
listfile = ''
test = 'N'

call initialize

call PARTS

exit

/******
/*      Functions      */
/******

PARTS :

/* Check if we have a 'listfile' or not as one of the parameters */
if listfile = ' ' then do /* If not... */

    command = 'Part -checkout '||name||' -family '||family||' -workarea '||workarea||' -release '||release||become||targetdir||' -verbose'

    call runCmd command

end /* Do */
else do /* If we have, we come here */

end /* do */
i = 0
do while lines(listfile) /* read in the listfile */

    i = i + 1
    log.i = linein(listfile)
    fullfile = log.i
    name = filespec('NAME',fullfile)

    targetdrive= filespec('DRIVE',fullfile)

    if targetdrive <> ' ' then do
        targetdir= filespec('PATH',fullfile)
        targetdir = ' -relative '||targetdrive||targetdir
    end /* do */

```

```

        command = 'Part -checkout '||name||' -family '||family||' -workarea '||workarea||' -release '||release||become||targetdir||' -verbose'

        call runCmd command
    end /* do */
    Myrc=stream(listfile,'c','close')    /* close the listfile */
end /* Do */

report = 'Report -view partfullview -family '||family||' -where "leasename = ''',
        ||release||'''' -verbose'

call runCmd report

return

/*****
/* Process input parameters and initialize variables */
*****/

initialize:

say 'input_string =' input_string
do forever while input_string <> ""
    parse var input_string next input_string
    parse upper var next next dummy

    select
        when next = '-B' then
            call BECOME
        when next = '-F' then
            call FAMILY
        when next = '-L' then
            call LISTFILE
        when next = '-N' then
            call NAME
        when next = '-R' then
            call RELEASE
        when next = '-T' then
            call TARGET
        when next = '-D' then
            call TEST
        when next = '-W' then
            call WORKAREA
        otherwise
            call HELP

    end /* select */
end /* do */

/* Check input parameters for values if none set default */

if family = '' then
    family = value('TC_FAMILY',,'OS2ENVIRONMENT')
if name = '' then
    name = '*.*'
if become = '' then
    become = value('TC_BECOME',,'OS2ENVIRONMENT')
if release = '' then
    release = value('TC_RELEASE',,'OS2ENVIRONMENT')
if targetdir = '' then
    targetdir = value('TC_TARGET',,'OS2ENVIRONMENT')
if workarea = '' then
    workarea = value('TC_WORKAREA',,'OS2ENVIRONMENT')

/* Display what we got and ask user for Y or N */

say 'name      =' name
say 'family    =' family
say 'release   =' release
say 'workarea  =' workarea
say 'become    =' become
say 'targetdir =' targetdir
say 'listfile  =' listfile
say 'test run  =' test
say ' '
say 'Correct ? Y/N '

parse upper pull answer

if answer = 'Y' then
    nop
else
    exit 0

```

```

/* If some of the mandatory parameters are missing, call help */

if family = '' | name = '' | release = '' | workarea = '' then
    call help

if become <> ' ' then
    become = ' -become '||become

if targetdir <> ' ' then
    targetdir = ' -relative '||targetdir

/* setup the log */
logfile = 'tcchkout.log'
'@copy tcchkout.log tcchkout.lo2 1>NUL 2>NUL'
'@del tcchkout.log 1>NUL 2>NUL'

return

/*****/
/* check that variables are set */
/*****/

FAMILY:
    parse var input_string next input_string
    if next <> '' then
        family = next
    else
        call help
return

NAME:
    parse var input_string next input_string
    if next <> '' then
        name = next
    else
        call help
return

BECOME:
    parse var input_string next input_string
    if next <> '' then
        become = next
    else
        call help
return

RELEASE:
    parse var input_string next input_string
    if next <> '' then
        release = next
    else
        call help
return

WORKAREA:
    parse var input_string next input_string
    if next <> '' then
        workarea = next
    else
        call help
return

TARGET:
    parse var input_string next input_string
    if next <> '' then
        targetdir = next
    else
        call help
return

LISTFILE:
    parse var input_string next input_string
    if next <> '' then
        listfile = next
    else
        call help
return

TEST:
    test = 'Y'
return

```

```

/*****
/*          runCmd          */
/*****
/* Executes a command and logs the results to a the */
/* 'tcchkout.log' file */
/*          */
/* It will also beep if there is an unexpected result. */
/*****
runCmd:

    parse arg  cmd

    cmd = 'teamc '||cmd
    call lineout logfile, ' '
    call lineout logfile

    /* log and run the command */
    "@echo === " cmd
    "@echo === " cmd ">>" logfile
    if test = 'N' then do
        "@cmd "1>buf.out 2> buf.err"
        saverc = rc

        /* check if result is expected */
        if saverc <> 0 then do
            BEEP(440,200)

            /* type results to the screen */
            "@echo --- rc=" saverc
            "@type  buf.out"
            "@type  buf.err"
        end /* do */

        /* type results to the logfile */
        "@echo --- rc=" saverc ">>" logfile
        "@type buf.out >>" logfile
        "@type buf.err >>" logfile

    end /* Do */

    call lineout logfile, " "
    call lineout logfile

return

help:

say '-----'
say '*'          TEAMC OS/2 Server Parts check-out          '*'
say '*' This Rexx program checks out the PARTS from your family. '*'
say '*' Author: Leif Trulsson, IBM ITSO San Jose - The Redbook People '*'
say '-----'
say' SYNTAX: TCCHKOUT -n name -f family -r release -w workarea '
say'                  <-t targetdir -b become -l listfile -d>      '
say' Where :
say'   -n name          = Specifies the parts to be checked out. Can be
say'                     specified in a couple of formats:
say'                     Unique name - check out part corresponding to given
say'                     name
say'                     Generic name - check out part(s) corresponding either
say'                     to files in path given or names
say'                     specified in the listfile.
say'                     If listfile is specified, only needs
say'                     to be specified.
say'   -f family        = Name of the family were the parts are stored.
say'   -r release        = The name of the release.
say'   -w workarea       = The name of the work area.
say'   -t targetdir      = The name of the target directory.
say'   -l listfile       = File containing list of parts to be checked out.
say'   -b become         = The TC_BECOME user ID.
say'   -d               = Debug test mode, does not run the command.
say' PRESS ENTER FOR MORE...'
parse pull dummy
say' Results of the program are written to the file TCCHKOUT.LOG'
say' The program also creates a partfullview Report based on family and
say' release name.'
exit 0

```


Figure 191. The tcchkout.cmd Program

C.5 A Program to Create Multiple Parts

This REXX programs enables you to create multiple parts at the same time. Figure 192 shows the *tcparts.cms* REXX program.

```

+-----+
/*****
/*      TEAMC OS/2 Part Create procedure      */
/*****
/* This REXX program creates PART(S) in given workarea */
/* and Release, connected to given component, and      */
/* fetched from either the current directory or the    */
/* path specified. The given path\file name can be     */
/* generic.                                           */
/*                                                    */
/* It is broken up into 3 sections.                  */
/* 1. TEAMC initialization                            */
/* 2. TEAMC Create PARTS                             */
/* 3. TEAMC Report on PARTS                           */
/*                                                    */
/*****
/* Last update: 01/11/96
/* Author      : Leif Trulsson, IBM ITSO San Jose
/*****
call RxFuncAdd 'SysCls' , 'RexxUtil', 'SysCls'
call RxFuncAdd 'SysTextScreenSize' , 'RexxUtil', 'SysTextScreenSize'
call RxFuncAdd 'SysCurPos' , 'RexxUtil', 'SysCurPos'
call RxFuncAdd 'SysFileTree' , 'RexxUtil' , 'SysFileTree'

env='OS2ENVIRONMENT'

parse arg input_string

/* Set up common variables */

family = ''
path = ''
comp = ''
release = ''
workarea = ''
listfile = ''
type = '-text'
test = 'N'

call initialize

call PARTS

exit

/*****
/*      Functions
/*****

PARTS:

/* Check if we have a 'listfile' or not as oner of the parameters */
if listfile = ' ' then do /* If not... */

    rc = SysFileTree(path , 'filelist' , 'FO')
    do i = 1 to filelist.0
        fullfile = filelist.i
        name = filespec('NAME',fullfile)
        say 'Part name =' name

        command = 'Part -create '||name||' -from '||fullfile||' -family '||family||' -workarea ',
                    ||workarea||' -component '||comp||' -release',
                    ||release type||' -verbose'

        call runCmd command

    end
end /* Do */
else do /* If we have, we come her */
    i = 0
    do while lines(listfile) /* read in the listfile */
        i = i + 1
        log.i = linein(listfile)
        fullfile = log.i
        name = filespec('NAME',fullfile) /* Get the file name */
        say 'Part name =' name
        command = 'Part -create '||name||' -from '||fullfile||' -family '||family||' -workarea ',

```

TeamConnection at Large A Program to Create Multiple Parts

```

||workarea||' -component '||comp||' -release|',
||release type||' -verbose'

    call runCmd command
end /* do */
Myrc=stream(listfile,'c','close')    /* close the listfile          */
end /* Do */

report = 'Report -view partfullview -family '||family||' -where "eleasename = "',
        ||release||'" -verbose'

call runCmd report

return

/*****/
/* Process input parameters and initialize variables */
/*****/
initialize:

say 'input_string =' input_string
do forever while input_string <> ""
    parse var input_string next input_string
    parse upper var next next dummy

    select
        when next = '-F' then
            call family
        when next = '-P' then
            call path
        when next = '-C' then
            call comp
        when next = '-D' then
            call TEST
        when next = '-B' then
            call TYPE
        when next = '-L' then
            call LISTFILE
        when next = '-R' then
            call release
        when next = '-W' then
            call workarea
        otherwise
            call help

    end /* select */
end /* do */

/* Check input parameters for values if none set default */

if family = '' then
    family = value('TC_FAMILY',,, 'OS2ENVIRONMENT')
if path = '' then
    path = value('TC_PATH',,, 'OS2ENVIRONMENT')
    if path = '' then
        path = '*.*'
if comp = '' then
    comp = value('TC_COMP',,, 'OS2ENVIRONMENT')
if release = '' then
    release = value('TC_RELEASE',,, 'OS2ENVIRONMENT')
if workarea = '' then
    workarea = value('TC_WORKAREA',,, 'OS2ENVIRONMENT')

/* Display what we got and ask user for Y or N */

say 'family      =' family
say 'path        =' path
say 'component    =' comp
say 'release      =' release
say 'workarea     =' workarea
say 'listfile     =' listfile
say 'part type    =' type
say 'test run     =' test
say ' '
say 'Correct ? Y/N '

parse upper pull answer

if answer = 'Y' then
    nop
else
    exit 0

/* If some of the mandatory parameters are missing, call help */

```

```

if family = '' | path = '' | comp = '' | release = '' | workarea = '' then
    call help

/* setup the log */
logfile = 'tcparts.log'
'@copy tcparts.log tcparts.lo2 1>NUL 2>NUL'
'@del tcparts.log 1>NUL 2>NUL'

return

/*****
/* check that variables are set */
*****/

family:
    parse var input_string next input_string
    if next <> '' then
        family = next
    else
        call help
return

path:
    parse var input_string next input_string
    if next <> '' then
        path = next
    else
        call help
return

comp:
    parse var input_string next input_string
    if next <> '' then
        comp = next
    else
        call help
return

release:
    parse var input_string next input_string
    if next <> '' then
        release = next
    else
        call help
return

workarea:
    parse var input_string next input_string
    if next <> '' then
        workarea = next
    else
        call help
return

LISTFILE:
    parse var input_string next input_string
    if next <> '' then
        listfile = next
    else
        call help
return

TYPE:
    type = '-binary'
return

TEST:
    test = 'Y'
return

/*****
/*                               */
/*      runCmd                  */
*****/
/* Executes the command and logs the results to the      */
/* 'tcparts.log' file                                     */
/*                               */
/* It will also beep if there is an unexpected result. */
*****/
runCmd:

    parse arg  cmd

```

```

cmd = 'teamc '||cmd
call lineout logfile, ' '
call lineout logfile

/* log and run the command */
"@echo === " cmd
"@echo === " cmd ">>" logfile
if test = 'N' then do
    "@cmd "1>buf.out 2> buf.err"
    saverc = rc

    /* check if result is expected */
    if saverc <> 0 then do
        BEEP(440,200)

        /* type results to the screen */
        "@echo --- rc=" saverc
        "@type buf.out"
        "@type buf.err"
    end /* do */

    /* type results to the logfile */
    "@echo --- rc=" saverc ">>" logfile
    "@type buf.out >>" logfile
    "@type buf.err >>" logfile

end /* Do */

call lineout logfile, " "
call lineout logfile

return

help:

say' '
say' *-----*-'
say' *          TEAMC OS/2 Server Parts create          *-'
say' * This Rexx program creates the PARTS for your installation. *-'
say' * Author: Leif Trulsson, IBM ITSO San Jose - The Redbook People *-'
say' *-----*-'
say' SYNTAX: TCPARTS -p path -f family -r release -c component -w workarea '
say'                  <-b -l listfile -d> '
say' Where :
say'   -p path          = Specifies source file name in the following form:
say'                   <d:\path\>filename      where: Target part name   filename'
say'                   Path can also be in a generic form like: '
say'                   *.c , *.* , or d:\path\*. * and the program will then '
say'                   create parts from all files satisfying the criteria.'
say'   -f family        = Name of target family where the parts are to be stored.'
say'   -c component      = The name of the target component.'
say'   -r release        = The name of the target release.'
say'   -w workarea       = The name of the target workarea.'
say'   -l listfile       = File containing list of parts to be created.'
say'   -b               = Part will be created with -binary (-text default).'
say'   -d               = Debug test mode, does not run the command.'
say' '
say' Results of the program are written to the file TCPARTS.LOG'
say' The program also creates a partfullview Report based on family and '
say' release name.'
exit 0

```

Figure 192. The tcparts.cmd Program

C.6 A Program to Extract Multiple Parts

This REXX program extracts multiple parts. Figure 193 shows the `tcextract.cmd` REXX program.

```

+-----+
/*****
/*      TEAMC OS/2 Part Extract  program      */
/*****
/* This REXX program extracts PART(s) from a given      */
/* workarea, release and family to a specified directory*/
/*
/* It is broken up into 3 sections.
/* 1. TEAMC initialization
/* 2. TEAMC extract  PARTS
/* 3. TEAMC Report on PARTS
/*
/*****
/* Last update: 05/07/96
/* Author      : Leif Trulsson, IBM ITSO San Jose
/*****
call RxFuncAdd 'SysCls' , 'RexxUtil', 'SysCls'
call RxFuncAdd 'SysTextScreenSize' , 'RexxUtil', 'SysTextScreenSize'
call RxFuncAdd 'SysCurPos' , 'RexxUtil', 'SysCurPos'
call RxFuncAdd 'SysFileTree' , 'RexxUtil' , 'SysFileTree'

env='OS2ENVIRONMENT'

parse arg input_string

/* Set up common variables */

family = ''
name = ''
become = ''
release = ''
workarea = ''
targetdir = ''
listfile = ''
test = 'N'

call initialize

call PARTS

exit

/*****
/*  Functions
/*****

PARTS :
/* Check if we have a 'listfile' or not as one of the parameters */
if listfile = '' then do /* If not... */
    rc = SysFileTree(name , 'filelist' , 'FO')
    do i = 1 to filelist.0
        fullfile = filelist.i
        name = filespec('NAME',fullfile)
        say 'Part name =' name

        command = 'Part -extract '||name||' -family '||family||' -workarea '||workarea||' -release ' ,
                    ||release||become||targetdir||' -verbose'

        call runCmd command

    end
end /* Do */
else do /* If we have, we come here */
    i = 0
    do while lines(listfile) /* read in the listfile */
        i = i + 1
        log.i = linein(listfile)
        fullfile = log.i
        name = filespec('NAME',fullfile)
        say 'Part name =' name
        command = 'Part -extract '||name||' -family '||family||' -workarea '||workarea||' -release ' ,
                    ||release||become||targetdir||' verbose'

        call runCmd command
    end /* do */
    Myrc=stream(listfile,'c','close') /* close the listfile
end /* Do */

```

```

report = 'Report -view partfullview -family '||family||' -where "leasename = ''',
        ||release||'''" -verbose'

call runCmd report

return

/*****
/* Process input parameters and initialize variables */
*****/

initialize:

say 'input_string =' input_string
do forever while input_string <> ""
    parse var input_string next input_string
    parse upper var next next dummy

    select
        when next = '-B' then
            call BECOME
        when next = '-F' then
            call FAMILY
        when next = '-L' then
            call LISTFILE
        when next = '-N' then
            call NAME
        when next = '-R' then
            call RELEASE
        when next = '-S' then
            call SOURCE
        when next = '-D' then
            call TEST
        when next = '-W' then
            call WORKAREA
        otherwise
            call HELP

    end /* select */
end /* do */

/* Check input parameters for values if none set default */

if family = '' then
    family = value('TC_FAMILY','','OS2ENVIRONMENT')
if name = '' then
    name = '.*.*'
if become = '' then
    become = value('TC_BECOME','','OS2ENVIRONMENT')
if release = '' then
    release = value('TC_RELEASE','','OS2ENVIRONMENT')
if targetdir = '' then
    targetdir = value('TC_TOP','','OS2ENVIRONMENT')
if workarea = '' then
    workarea = value('TC_WORKAREA','','OS2ENVIRONMENT')

/* Display what we got and ask user for Y or N */

say 'name      =' name
say 'family    =' family
say 'release   =' release
say 'workarea  =' workarea
say 'become    =' become
say 'targetdir =' targetdir
say 'test run  =' test
say ' '
say 'Correct ? Y/N '

parse upper pull answer

if answer = 'Y' then
    nop
else
    exit 0

/* If some of the mandatory parameters are missing, call help */

if family = '' | name = '' | release = '' | workarea = '' then
    call help

if become <> ' ' then
    become = ' -become '||become

```

```

if targetdir <> ' ' then
    targetdir = ' -relative '||targetdir

/* setup the log */
logfile = 'tcxtract.log'
'@copy tcxtract.log tcxtract.lo2 1>NUL 2>NUL'
'@del tcxtract.log 1>NUL 2>NUL'

return

/*****/
/* check that variables are set */
/*****/

FAMILY:
    parse var input_string next input_string
    if next <> ' ' then
        family = next
    else
        call help
return

NAME:
    parse var input_string next input_string
    if next <> ' ' then
        name = next
    else
        call help
return

BECOME:
    parse var input_string next input_string
    if next <> ' ' then
        become = next
    else
        call help
return

RELEASE:
    parse var input_string next input_string
    if next <> ' ' then
        release = next
    else
        call help
return

WORKAREA:
    parse var input_string next input_string
    if next <> ' ' then
        workarea = next
    else
        call help
return

TARGET:
    parse var input_string next input_string
    if next <> ' ' then
        targetdir = next
    else
        call help
return

LISTFILE:
    parse var input_string next input_string
    if next <> ' ' then
        listfile = next
    else
        call help
return

TEST:
    test = 'Y'
return

/*****/
/*                               */
/*          runCmd                */
/*****/
/* Executes a command and logs the results to the */
/* 'tcxtract.log' file                      */
/*                               */
/* It will also beep if there is an unexpected result. */
/*****/

```



```

runCmd:

    parse arg  cmd

    cmd = 'teamc '||cmd
    call lineout logfile, ' '
    call lineout logfile

    /* log and run the command */
    "@echo === " cmd
    "@echo === " cmd ">>" logfile
    if test = 'N' then do
        "@cmd "1>buf.out 2> buf.err"
        saverc = rc

        /* check if result is expected */
        if saverc <> 0 then do
            BEEP(440,200)

            /* type results to the screen */
            "@echo --- rc=" saverc
            "@type  buf.out"
            "@type  buf.err"
        end /* do */

        /* type results to the logfile */
        "@echo --- rc=" saverc ">>" logfile
        "@type buf.out >>" logfile
        "@type buf.err >>" logfile

    end /* Do */

    call lineout logfile, " "
    call lineout logfile

return

help:

say '-----'
say '* TEAMC OS/2 Server Parts extract'
say '* This Rexx program extracts the PARTS from your family.'
say '* Author: Leif Trulsson, IBM ITSO San Jose - The Redbook People'
say '-----'
say' SYNTAX: TCXTRACT -n name -f family -r release -w workarea '
say'                  <-t targetdir -b become -l listfile -d>
say' Where : '
say'   -n name       = Specifies the parts to be extracted. Can be '
say'                  specified in a couple of formats:
say'                  Unique name - extract part corresponding to given '
say'                  name'
say'                  Generic name - extract part(s) corresponding either '
say'                  to files in path given or names '
say'                  specified in the listfile.'
say'                  If listfile is specified, only ' needs '
say'                  to be specified.'
say'   -f family     = Name of the family were the parts are stored.'
say'   -r release     = The name of the release.'
say'   -w workarea    = The name of the work area'
say'   -t targetdir   = The name of the target directory.'
say'   -b become      = The TC_BECOME user ID.'
say'   -l listfile    = File containing list of parts to be checked out.'
say'   -d             = Debug test mode, does not run the command.'
say' PRESS ENTER FOR MORE...'
parse pull dummy
say' Results of the program are written to the file TCXTRACT.LOG'
say' The program also creates a partfullview Report based on family and '
say' release name.'
exit 0

```

Figure 193. The tcxtract.cmd Program

D.0 Appendix D. Special Notices

This publication is intended to help application development managers, project leaders, product evaluators, service providers, family administrators, system administrators, developers, and team leaders understand how to plan for, install, use, and take advantage of TeamConnection in regard to the company's development efforts. The information in this publication is not intended as the specification of any programming interfaces that are provided by TeamConnection. See the PUBLICATIONS section of the IBM Programming Announcement for TeamConnection for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AD/Cycle	AIX
AIX/6000	C Set ++
C/MVS	C/370
CMVC	COBOL/2
COBOL/370	CUA
IBM	MVS
NVBridge/2	NetView
OS/2	Presentation Manager
PS/2	REXX
TeamConnection	VisualAge
VisualAge C++	VisualAge for COBOL
VisualGen	Workplace Shell

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

ObjectStore	Object Design Inc.
OSF/Motif	Open Software Foundation

Other trademarks are trademarks of their respective companies.

E.0 Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

Subtopics

E.1 International Technical Support Organization Publications

E.2 Other Publications

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications, see "How to Get ITSO Redbooks" in topic BACK_1:

- ☐ *Introduction to the IBM Application Development Team Suite, SG24-4648*
- ☐ *Did You Say CMVC?, GG24-4178*
- ☐ *TeamConnection and WorkFrame Integration Survival Guide, SG24-4610*

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

E.2 Other Publications

These publications are also relevant as further information sources:

Subtopics

E.2.1 TeamConnection

E.2.2 NetView DM/2

E.2.3 ObjectStore Release 4.0 Publications

E.2.4 IBM OS/2 Publications

E.2.1 TeamConnection

- ☐ *IBM TeamConnection for OS/2 Getting Started, SC34-4498*
- ☐ *IBM TeamConnection for OS/2 User's Guide, SC34-4499*
- ☐ *IBM TeamConnection for OS/2 Commands Reference, SC34-4501*
- ☐ *IBM TeamConnection for OS/2 Messages, SC34-4502*

E.2.2 NetView DM/2

- ☐ *NetView DM/2 Concepts and Overview, GH19-4009*
- ☐ *NetView DM/2 CDM User's Guide, SH19-5048*
- ☐ *NetView DM/2 LDU User's Guide, SH19-5049*
- ☐ *NetView Distribution Manager/2 Messages and Error Recovery Guide, SH19-6924*

E.2.3 ObjectStore Release 4.0 Publications

- ☐ *ObjectStore Management, 310-000-40M*
- ☐ *ObjectStore C++ Installation, 310-320-40I*
- ☐ *ObjectStore C++ Release Notes, 310-000-40N*

E.2.4 IBM OS/2 Publications

- ☐ *REXX User's Guide*, S10G-6269
- ☐ *REXX Reference*, S10G-6268

BACK_1 How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL **<http://www.redbooks.ibm.com/redbooks>**.

Subtopics

BACK_1.1 How IBM Employees Can Get ITSO Redbooks

BACK_1.2 How Customers Can Get ITSO Redbooks

BACK_1.3 IBM Redbook Order Form

BACK_1.1 How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

☐ **PUBORDER** -- to order hardcopies in United States

☐ **GOPHER link to the Internet**

Type **GOPHER.WTSCPOK.ITSO.IBM.COM**

☐ **Tools disks**

To get LIST3820s of redbooks, type one of the following commands:

```
TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
```

To get lists of redbooks:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
```

To register for information on workshops, residencies, and redbooks:

```
TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
```

For a list of product area specialists in the ITSO:

```
TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
```

☐ **Redbooks Home Page on the World Wide Web**

<http://w3.itso.ibm.com/redbooks/redbooks.html>

☐ **IBM Direct Publications Catalog on the World Wide Web**

<http://www.elink.ibm.link.ibm.com/pbl/pbl>

IBM employees may obtain LIST3820s of redbooks from this page.

☐ **ITSO4USA category on INEWS**

☐ **Online** -- send orders to:

USIB6FPL at IBMAIL or DKIBMBSH at IBMAIL

☐ **Internet Listserver**

With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to **announce@webster.ibm.link.ibm.com** with the keyword **subscribe** in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

BACK_1.2 How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- ☐ **Online Orders** (Do not send credit card information over the Internet)

IBMMAIL -- send orders to:

In United States:	usib6fpl at ibmmail
In Canada:	caibmbkz at ibmmail
Outside North America:	bookshop at dkibmbsh at ibmmail

Internet -- send orders to:

In United States:	usib6fpl@ibmmail.com
In Canada:	lmannix@vnet.ibm.com
Outside North America:	bookshop@dk.ibm.com

- ☐ **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU

Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- ☐ **Mail Orders** -- send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29554	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- ☐ **Fax** -- send orders to:

United States (toll free)	1-800-445-9269
Canada (toll free)	1-800-267-4455
Outside North America	(+45) 48 14 2207
(long distance charge)	

- ☐ **1-800-IBM-4FAX (United States) or (+1) 415 855 43 29 (Outside USA)**

Ask for:

- Index # 4421 Abstracts of new redbooks
- Index # 4422 IBM redbooks
- Index # 4420 Redbooks for last six months

- ☐ **Direct Services**

Send note to **softwareshop@vnet.ibm.com**

- ☐ **Redbooks Home Page on the World Wide Web**

<http://www.redbooks.ibm.com/redbooks>

- ☐ **IBM Direct Publications Catalog on the World Wide Web**

<http://www.elink.ibm.link.ibm.com/pbl/pbl>

- ☐ **Internet Listserver**

With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to **announce@webster.ibm.link.ibm.com** with the keyword **subscribe** in the body of the note (leave the subject line blank).

BACK_1.3 IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quant

Please put me on the mailing list for updated versions of the IBM Redbook Catalog.

First name

Last name

Company

Address

City

Postal code

Country

Telephone number

Telefax number

VAT number

Invoice to customer number

Credit card number

Credit card expiration date

Card issued to

Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

GLOSSARY Glossary

This glossary defines the terms and abbreviations used in this book. If you do not find the term you are looking for, refer to the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

```
+---+
|  A  |
+---+
```

absolute path name. A directory or a part expressed as a sequence of directories followed by a part name beginning from the root directory.

access list. A set of objects that controls access to data. Each object consists of a component, a user, and the authority that the user is granted or is restricted from in that component. See also *authority* and *restricted authority*.

action. A task performed by the TeamConnection server and requested by a TeamConnection client. A TeamConnection action is the same as issuing one TeamConnection command.

agent. See *build agent*.

alternate version ID. In collision records, the name of a version of a driver, release, or work area where the conflicting version of a part is visible.

approval record. A status record on which an approver must give an opinion of the proposed part changes required to resolve a defect or implement a feature in a release.

approver. A user who has the authority to mark an approval record with accept, reject, or abstain within a specific release.

approver list. A list of user IDs attached to a release, representing the users who must review part changes that are required to resolve a defect or implement a feature in that release.

attribute. Information contained in a field that is accessible to the user. TeamConnection enables family administrators to customize defect, feature, user, and part tables by adding new attributes.

authority. The right to access development objects and perform TeamConnection commands. See also *access list*, *base authority*, *explicit authority*, *implicit authority*, and *restricted authority*.

```
+---+
|  B  |
+---+
```

base authority. The set of actions granted to a user when a user ID is created within a TeamConnection family. See also *authority*. Contrast with *implicit authority* and *explicit authority*.

build. The process used to create applications within TeamConnection.

build agent. A program that handles access to persistent data on behalf of the build processor. Each build agent is connected to one and only one build processor, through a TCP/IP connection.

build cache. A directory that the build processor uses to enhance performance.

build dependent. A TeamConnection part that is needed for the compile operation to complete but will not be passed directly to the compiler. An example of this is an include file. See also *dependencies*.

builder. An object that can transform one set of TeamConnection parts into another by invoking tools such as compilers and linkers.

build event. An individual step in the build of an application, such as the compiling of **hello.c** into **hello.obj**.

build input. A TeamConnection part that will be used as input to the object being built.

build output. A TeamConnection part that will be generated as output from a build, such as an **.obj** or **.exe** file.

build pool. A group of build servers that resides in an environment. The environment in which several build servers operate. Typically, several servers are set up for each environment for which the enterprise develops

applications.

build processor. A program that invokes tools, such as compilers and linkers, that construct an application. Each build processor is connected to one and only one build agent, through a TCP/IP connection. See also *build agent* and *build cache*.

build scope. A collection of build events that implement a specific build request. See also *build event*.

build script. An executable or command file that specifies the steps that should occur during a build operation. This file can be a compiler, a linker, or the name of a *.cmd* file you have written.

build server. The combination of a build processor and a build agent. See also *build agent* and *build processor*.

build target. The name of the part at the top of the build tree that is the final output of a build. TeamConnection uses the build target to determine the scope of the build. See also *build tree*.

build tree. A graphical representation of the dependencies that the parts in an application have on one another. If you change the relationship of one part to another, the build tree changes accordingly.

```
+---+
|  C  |
+---+
```

change control process. The process of limiting and auditing changes to parts through the mechanism of checking parts in and out of a central, controlled, storage location. Change control for individual releases can be integrated with problem tracking by specifying a process for the release that includes the tracking subprocess.

check-in. The return of a TeamConnection part to version control.

check-out. The retrieval of a version of a part under TeamConnection control. In nonconcurrent releases, the check-out operation does not allow a second user to check out a part until the first user has checked it back in.

child component. Any component in a TeamConnection family, except the root component, that is created in reference to an existing component. The existing component is the parent component, and the new component is the child component. A parent component can have more than one child component, and a child component can have more than one parent component. See also *component* and *parent component*.

child part. Any part in a build tree that has a parent defined. A child part can be input, output, or dependent. See also *part* and *parent part*.

client. A functional unit that receives shared services from a server. Contrast with *server*.

collision record. A status record associated with a work area or driver, a part, and one of the following:

- ☐ The work area or driver's release
- ☐ Another work area

TeamConnection generates a collision record when a changed version of a part conflicts with a previously committed and integrated version of the same part. This is only related to *concurrent* development mode, when more than one developer can check out the same version of the same part concurrently. (In serial development, the part will be locked after the first check-out.)

command. A request to perform an operation or run a program from the command line interface. In TeamConnection, a command consists of the command name, one action flag, and zero or more attribute flags.

command line. (1) An area on the Tasks window or in the TeamConnection Commands window where a user can type TeamConnection commands. (2) An area on an OS/2 window where you can type TeamConnection commands.

committed version. The revision of a part that is visible from the release.

common part. A part that is shared by two or more releases, and the same version of the part is the current version for those releases.

comparison operator. An operator used in comparison expressions. Comparison operators used in TeamConnection are > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), and = (equal to).

component. A TeamConnection object that organizes project data into structured groups and controls configuration management properties. Component owners can control access to data and notification of TeamConnection actions. Components exist in a parent-child hierarchy with descendant components inheriting access and notification information from ancestor components. See also *access list* and *notification list*.

concurrent development. Several users can work on the same part at the same time. TeamConnection requires these users to reconcile their changes when they commit or integrate their work areas and drivers with the release. Contrast with *serial development*. See also *work area*.

configuration management. The process of identifying, managing, and controlling software modules as they change over time.

connect. The process of linking parts so that they are included in a build.

context. The current work area or driver used for part operations.

corequisite work areas. Two or more work areas designated as corequisites by a user so that all work areas in the corequisite group must be included as members in the same driver, before that driver can be committed. If the driver process is not used in the release, corequisite work areas must be integrated by the same command. See also *prerequisite work areas*.

current version. The last visible modification of a part in a driver, release, or work area.

current working directory. (1) The directory that is the starting point for relative path names. (2) The directory in which you are working.

```
+---+
|  D  |
+---+
```

daemon. A program that runs unattended to perform a standard service. Some daemons are triggered automatically to perform their task; others operate periodically.

database. A collection of data that can be accessed and operated by a data processing system for a specific purpose.

default. A value that is used when an alternative is not specified by the user.

default query. A database search, defined for a specific TeamConnection window, that is issued each time that TeamConnection window is opened. See also *search*.

defect. A TeamConnection object used to formally report a problem. The user who opens a defect is the defect originator.

delete. If you delete a development object, such as a part or a user ID, any reference to that object is removed from TeamConnection. Certain objects can be deleted only if certain criteria are met. Most objects that are deleted can be re-created.

delta part tree. A directory structure representing only the parts that were changed in a specified place.

dependencies. In TeamConnection builds there are two types of dependencies:

- ☐ **automatic.** These are build dependencies that a parser identifies.
- ☐ **manual.** These are build dependencies that a user explicitly identifies in a build tree.

See also *build dependent*.

destroy. To remove the part record from the database on the TeamConnection server. The only TeamConnection development object that can be destroyed is a part.

disconnect. The process of unlinking parts so that they are not included

in a build.

driver. A collection of work areas that represent a set of changed parts within a release. Drivers are only associated with releases whose processes include the track and driver subprocesses.

driver member. A work area that is added to a driver.

```
+---+
| E |
+---+
```

environment. (1) A user-defined testing domain for a particular release. (2) A defect field representing the environment where the problem occurred. (3) The string that matches a build agent with a build event.

environment list. A TeamConnection object used to specify environments in which a release should be tested. A list of environment-user ID pairs attached to a release, representing the user responsible for testing each environment. Only one tester can be identified for an environment.

explicit authority. The ability to perform an action against a TeamConnection object because you have been granted the authority to perform that action. Contrast with *base authority* and *implicit authority*.

extract. A TeamConnection action you can perform on a part, driver, or release. A part extraction results in copying the specified part to a client workstation. A driver extraction and release extraction result in all parts for the driver or release being copied to a designated location.

```
+---+
| F |
+---+
```

family. A logical organization of related data. A single TeamConnection server can support multiple families. The data in one family cannot be accessed from another family.

family administrator. A user responsible for all non-system-related tasks for one or more TeamConnection families, such as planning, configuring, and maintaining the TeamConnection environment and managing user access to those families.

family server. A workstation running the TeamConnection server software.

feature. A TeamConnection object used to formally request and record information about a functional addition or enhancement. The user who opens a feature is the feature originator.

file allocation table (FAT). The DOS- and OS/2-compatible file system that manages input, output, and storage of files on your system. File names can be up to eight characters long, followed by a file extension that can be up to three characters.

fix record. A status record associated with a work area and that is used to monitor the phases of change within each component affected by a defect or feature for a specific release.

freeze. The freeze command saves changed parts to the work area. Thus, TeamConnection takes a snapshot of the work area, including all of the current versions of parts visible from that work area, and saves this image of the system. The user can always come back to this stage of development in the work area. Note, however, that a freeze action does not make the changes visible to other users working in the release.

full part tree. A directory structure representing a complete set of active parts associated with the release.

```
+---+
| G |
+---+
```

graphical user interface (GUI). A type of computer interface consisting of a visual metaphor of a real-world scene, often as a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

```
+---+
| H |
+---+
```

high-performance file system (HPFS). In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the existence of multiple, active file systems on a single personal computer, with the capacity of multiple and different storage devices. File names used with HPFS can have as many as 254 characters.

host. A host node, host computer, or host system.

host list. A list associated with each TeamConnection user ID that indicates the client machine that can access the family server and act on behalf of the user. The family server uses the list to authenticate the identity of a client machine when the family server receives a command. Each entry consists of a login, a host name, and a TeamConnection user ID.

host name. The identifier associated with the host computer.

```
+---+
| I |
+---+
```

implicit authority. The ability to perform an action on a TeamConnection object without being granted explicit authority. This authority is automatically granted through inheritance or object ownership. Contrast with *base authority* and *explicit authority*.

import. To bring in data. In TeamConnection, to bring selected items into a field from a matching TeamConnection object window.

inheritance. The passing of configuration management properties from parent to child component. The configuration management properties that are inherited are access and notification. Inheritance within each TeamConnection family or component hierarchy is cumulative.

integrated problem tracking. The process of integrating problem tracking with change control to track all reported defects, proposed features, and subsequent changes to parts. See also *change control process*.

interest group. The list of actions that trigger notification to the user IDs associated with those actions listed in the notification list.

```
+---+
| J |
+---+
```

job queue. A queue of build scopes. One job queue exists for each TeamConnection family.

```
+---+
| L |
+---+
```

lock. An action that prevents editing access to a part stored in the TeamConnection development environment so that only one user can change a part at a time.

login. The name that identifies a user on a multiuser system. In OS/2, the login value is obtained from the TC_USER environment variable.

```
+---+
| M |
+---+
```

metadata. In databases, data that describe data objects.

```
+---+
| N |
+---+
```

name server. In TCP/IP, a server program that supplies name-to-address translation by mapping domain names to Internet addresses.

notification list. An object that enables component owners to configure notification. A list attached to a component that pairs a list of user IDs and a list of interest groups. It designates the users and the corresponding notification interest that they are being granted for all objects managed by this component or any of its descendants.

notification server. A server that sends notification messages to the client.

```
+---+
|  O  |
+---+
```

operator. A symbol that represents an operation to be done. See also *comparison operator*.

originator. The user who opens a defect or feature and is responsible for verifying the outcome of the defect or feature on a verification record. This responsibility can be reassigned.

owner. The user responsible for a TeamConnection object within a TeamConnection family, either because the user created the object or was assigned ownership of the object.

```
+---+
|  P  |
+---+
```

parent component. All components in each TeamConnection family, except the root component, are created in reference to an existing component. The existing component is the parent component. See also *child component* and *component*.

parent part. Any part in a build tree that has a child defined. See also *part* and *child part*.

parser. A tool that can read a source file and report back a list of dependencies of that source file. It frees a developer from knowing the dependencies one part has on other parts to ensure that a complete build is performed.

part. A collection of data that is stored by the family server and retrieved by a path name. Parts can be text objects, binary objects, and modeled objects. These parts can be stored by the user or the tool, or they can be generated from other parts, such as when a linker generates an executable file.

path name. The name of the part under TeamConnection control. A path name can be a directory structure and a base name or just a base name. It must be unique within each release.

prerequisite work areas. If a part is changed to resolve more than one defect or feature, the work area referenced by the first change is a prerequisite of the work area referenced by later changes. A work area is a prerequisite to another work area if:

- ☐ Part changes are checked in, but not committed, for the first work area.
- ☐ One or more of the same parts are checked out, changed, and checked in again for the second work area.

problem tracking. The process of tracking all reported defects through to resolution and all proposed features through to implementation.

process. A combination of TeamConnection subprocesses, configured by the family administrator, that controls the general movement of TeamConnection objects (defects, features, work areas, and drivers) from state to state within a component or release. See also *subprocess* and *state*.

```
+---+
|  Q  |
+---+
```

query. A request for information from a database, for example, a search for all defects that are in the open state. See also *default query* and *search*.

```
+---+
|  R  |
+---+
```

raw format. Information retrieved on the Report command that has the vertical bar delimiter separating field information, and each line of output corresponds to one database record.

refresh. This TeamConnection command updates a work area with any changes from the release. It also freezes the work area, if it is not already frozen.

relative path name. The name of a directory or a part expressed as a sequence of directories followed by a part name, beginning from the current directory.

release. A TeamConnection object defined by a user that contains all parts that must be built, tested, and distributed as a single entity.

restricted authority. The limitation on a user's ability to perform certain actions at a specific component. Authority can be restricted by the superuser, the component owner, or a user with AccessRestrict authority. See also *authority*.

root component. The initial component created when a TeamConnection family is configured. All components in a TeamConnection family are descendants of the root component. Only the root component has no parent component. See also *component*, *child component*, and *parent component*

```
+---+
| S |
+---+
```

search. To scan one or more data elements of a set in a database to find elements that have certain properties.

serial development. While a user has parts checked out from a work area, no one else on the team can check out the part. The user develops new material without interacting with other developers on the project. TeamConnection provides the opportunity to hold the part until the user is sure that it integrates with the rest of the application. Thus, the lock is not released until the work area as a whole is committed. Contrast with concurrent development. See also *work area*.

server. A workstation that performs a service for another workstation.

shell script. A series of commands combined in a file that carry out a function when the file is run.

sizing record. A status record created for each component-release pair affected by a proposed defect or feature. The sizing record owner must indicate whether the defect or feature affects the specified component-release pair and the approximate amount of work needed to resolve the defect or implement the feature within the specified component-release pair.

stanza format. Data output generated by the Report command in which each database record is a stanza. Each stanza line consists of a field and its corresponding values.

state. Work areas, drivers, features, and defects move through various states during their life cycles. The state of an object determines the actions that can be performed on it. See also *process* and *subprocess*.

subprocess. TeamConnection subprocesses govern the state changes for TeamConnection objects. The design, size, review (DSR) and verify subprocesses are configured for component processes. The track, approve, fix, driver, and test subprocesses are configured for release processes. See also *process* and *state*.

superuser. This privilege lets a user perform any action available in the TeamConnection family.

system administrator. A user who is responsible for all system-related tasks involving the TeamConnection server, such as installing, maintaining, and backing up the TeamConnection server and the database it uses.

```
+---+
| T |
+---+
```

task list. The list of tasks displayed in the Tasks window. The user can customize this list to issue requests for information from the server. Tasks can be added, modified, or deleted from the lists.

TeamConnection client. A workstation that connects to the TeamConnection server by a TCP/IP connection and that is running the TeamConnection client software.

TeamConnection part. A part that is stored by the TeamConnection server and retrieved by a path name, release, type, and work area. See also *part*, *common part*, and *type*.

TeamConnection superuser. See *superuser*.

tester. A user responsible for testing the resolution of a defect or the implementation of a feature for a specific driver of a release and recording the results on a test record.

test record. A status record used to record the outcome of an environment test performed for a resolved defect or an implemented feature in a specific driver of a release.

track subprocess. An attribute of a TeamConnection release process that specifies that the change control process for that release will be integrated with the problem tracking process.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

type. All parts that are created through the TeamConnection GUI or on the command line will show up in reports with the type of *file* as the part type. The TeamConnection GUI and command line can only check in, check out, and extract parts of the *type* file.

Note: Parts created through an API can have other specified types.

```
+---+
|  U  |
+---+
```

user exit. A user exit allows TeamConnection to call a user-defined program during the processing of TeamConnection transactions. User exits provide a means by which users can specify additional actions that should be performed before completing or proceeding with a TeamConnection action.

user ID. The identifier assigned by the system administrator to each TeamConnection user.

```
+---+
|  V  |
+---+
```

verification record. A status record that the originator of a defect or a feature must mark before the defect or feature can move to the closed state. Originators use verification records to verify the resolution or implementation of the defect or feature they opened.

version. (1) A specific view of a driver, release, or work area. (2) A revision of a part.

version control. The storage of multiple versions of a single part and information about each version.

view. An alternative and temporary representation of data from one or more tables.

```
+---+
|  W  |
+---+
```

work area. An object in TeamConnection that you create and associate with a release. When the work area is created, you see the most current view of the release and all the parts that it contains. You can check out the parts in the work area, make modifications, and check them back into the work area. You can also test the modifications without integrating them. Other users are not aware of the changes that you make in the work area until you integrate the work area to the release. While you work on files in a work area, you do not see subsequent part changes in the release until you integrate or refresh your work area.

working part. The checked-out version of a TeamConnection part.

ABBREVIATIONS List of Abbreviations

AD	application development
ANSI	American National Standards Institute
CMVC	Configuration Management Version Control
COBOL	Common Business Oriented Language
GUI	graphical user interface
HLL	high level language
IBM	International Business Machines Corporation
ISO	International Organization for Standardization
ITSO	International Technical Support Organization
LAN	local area network
MVS	Multiple Virtual Storage
OS/2	Operating System/2
REXX	Restructured Extended Executor Language
SEI	Software Engineering Institute
3GL	third generation language
VGA	video graphics array/adaptor
WA	work area
XGA	extended graphics adapter

A

- abstain 9.2.1 9.2.2.7 9.3.1
- accept 9.2.1 9.2.2.7 9.3.1 10.1.3
- accepting the defect 9.3.3.5
- accepting the feature 9.2.2.5
- access list 2.1 6.2.1
- action-state diagram 9.2.1 9.3.1 9.3.3 10.1.1 10.1.2
- add driver members 9.3.3.10
- add new processes 12.3.1
- adding a driver member 9.3.3.10
- aging utilities
 - age 11.1.1
 - resetage 11.1.2
- analyzing a defect or feature 9.2.1
- application development challenges 1.0 1.1
- approval record 9.3.1 9.4
- approver 9.3.3.7
- approver lists 9.3.1
- authority groups 5.2 9.1
- authority levels 3.5.1
- authorized transactions 11.2.2
- automated distribution 8.3

B

- backup and recovery 2.8
- base name 3.9 6.6.3
- bibliography E.0
- binary files 6.6.3
- break common link 6.7.2 6.7.3
- build administrator 7.0 7.1
- build agent 2.4.2 3.1.1 7.3 7.4
- build environment 3.8 7.3
- build event 2.4.1 7.7
- build events 7.3
- build pool 7.3
- build processor 2.4.2 3.1.1 7.4
- build scope 2.4.2
- build script 2.4.1 7.0 7.2
- build structure 2.4
- build topology 7.4
- build tree 2.4.1 7.0 7.7
- build tree for packaging 8.1
- build tree for the NVBridge/2 8.1.2
- builder 2.4.1 7.0 7.5
- builder parameters 7.2.1
- building an application 7.9

C

- change control 2.6 9.0 9.3 14.4.1.2
- change control process 9.0
- change existing processes 12.3.1
- change management 13.0 13.4
- change process 2.2
- changing report formats 12.5
- check in 6.7
- check out 6.7 6.7.1
- cleaning up the audit log 11.2.2.1
- coarse-grained parts 10.1
- collector object 7.8
- collision 10.1
- collision record 10.1.3
- collision records 10.1
- committing the driver 9.3.3.11
- common actions 2.1
- common releases 6.7.2 6.7.3
- complete fix records 9.3.3.9
- completing test records 9.3.3.13
- completing the driver 9.3.3.12
- completing the verification records 9.3.3.14
- component 2.1 6.3
- component hierarchy 2.1 3.4 3.4.1
- component process
 - planning 3.7.1
- component processes 5.5 9.0
- component structure 6.3.1
- composite file 10.1.4
- composite view 10.1.4
- conceptual view 2.9
- concurrent development 6.7.1 10.1 10.1.1 10.1.2 10.1.4
- configurable fields
 - add 5.4
 - changing 10.2 10.2.2
 - creating 10.2 10.2.2
 - defining types 10.2.1
 - displaying properties 10.2.3
 - update 5.4

- using 10.2
- configurable process 2.6
- Configuration Management 2.1 13.0 13.3 13.4
- configuring component processes 12.3
- configuring release processes 12.3
- control of nonconforming product 14.2 14.2.2.4
- create sizing records 9.2.2.3
- creating a build tree 7.7
- creating a builder 7.5
- creating a driver 9.3.3.10
- creating a parser 7.6
- creating a release 6.4
- creating authority groups 12.1
- creating configurable fields 12.4.2.1
- creating interest groups 12.2
- creating the family 5.0
- creating versions 6.8
- customer value 1.3

D

- data constraints 2.11
- defect 2.6 9.0
- defect and feature states
 - canceled state 9.2.1
 - closed 9.2.2.8 9.3.3.14
 - closed state 9.2.1
 - design 9.2.2.2
 - design state 9.2.1
 - open state 9.2.1
 - returned state 9.2.1
 - review 9.2.2.4
 - review state 9.2.1
 - size 9.2.2.3
 - size state 9.2.1
 - verify 9.2.2.7 9.3.3.13 9.3.3.14
 - verify state 9.2.1
 - working 9.2.2.6 9.3.3.6 9.3.3.14
 - working state 9.2.1
- defining configurable field types 12.4
- delta part tree 9.3.2 9.3.3.10
- design control 14.2 14.2.2.1
- design size review 9.2.1.1
- design text 9.2.2.4
- developer 7.1
- development cycle 3.6 9.3
- development methodologies 13.6
- development team 1.0
- distributed build 2.4.2
- distribution 8.0
- document control 14.2 14.2.1
- driver 9.3.2 9.3.3.10 9.3.3.11
- driver member 9.3.3.8
- driver members 9.3.2 9.3.3.10 9.3.3.11
- driver process 2.6
- driver states
 - commit 9.3.2 9.3.3.11
 - complete 9.3.2
 - integrate 9.3.2
 - working 9.3.2

E

- edit 6.7
- editing the authorit.ld file 12.1.1
- editing the comproc.ld file 12.3.1
- editing the interest.ld file 12.2.1
- editing the relproc.ld files 12.3.1
- electronic software distribution 2.5.2
- environment list 9.3.3.12 9.3.3.13
- Environment page 6.1.2.1
- environment variables
 - builder parameters 7.2.1
 - OS_NETWORK 4.4
 - setting 4.3
 - TC_BECOME 6.1.2.1
 - TC_CASESENSE 6.1.2.2
 - TC_COMPONENT 6.1.2.1
 - TC_FAMILY 6.1.2.1 7.2.1 12.3.2
 - TC_INPUT 7.2.1
 - TC_INPUTTYPE 7.2.1
 - TC_LOCATION 7.2.1
 - TC_NLSPATH 6.1.2.2
 - TC_OUTPUT 7.2.1
 - TC_OUTPUTTYPE 7.2.1
 - TC_RELEASE 6.1.2.1 7.2.1
 - TC_TOP 6.1.2.1
 - TC_USER 4.5.1 6.1.2.1

- TC_WORKAREA 6.1.2.1 7.2.1
- USER 4.1
- extended JCL syntax 7.2.2.1
- extract 6.7
- Extract page 6.1.2.4
- F**
 - family 2.1
 - family administrator 12.4
 - family planning 3.2
 - feature 2.6 9.0
 - fine-grained parts 10.1
 - fix record 9.4
 - fix records 9.2.2.3 9.3.1
 - formal testing 9.3.3.12
 - format specification 12.5.1
 - fsdata 8.3
 - full part tree 9.3.2 9.3.3.10 9.3.3.11
- G**
 - Gather/2 2.5.1 8.0 8.1.1
 - GUI page 6.1.2.3
- H**
 - hardware requirements 3.1.1
 - host list 3.3 6.2.1
 - host name 3.3
 - hosts file 4.1
- I**
 - IBM Netview Distribution Manager/2
 - See ?
 - impact analysis 2.7
 - information model 2.9 2.10 2.11 2.12
 - initial superuser 5.1
 - input managed objects 2.4.1
 - inspection and test status 14.2 14.2.2.3
 - installation options 4.2.2
 - installation planning 3.1
 - installing the TeamConnection components 4.2
 - integrated build 2.4
 - integrated problem tracking 2.6 9.0
 - integrating the work area 9.3.3.9
 - integration testing 9.3.3.10
 - interest groups 5.3 9.1
 - internal quality audits 14.2 14.2.3
- J**
 - job queue 2.4.2
- L**
 - lock 6.7
 - logical view 2.9
- M**
 - maintain a component structure 6.3.3
 - maintaining the TeamConnection database 11.3
 - managed object 2.1
 - managed objects 2.1
 - manually defining configurable field types 12.4.1
 - merge 10.1.3
 - merge tool 10.1 10.1.3
 - modeled objects 2.1
 - modifying authority groups 12.1
 - modifying interest groups 12.2
 - monitor the progress of a build 7.9
 - moving a defect to the design state 9.3.3.2
 - moving a defect to the review state 9.3.3.4
 - moving a defect to the size state 9.3.3.3
 - multiple families 3.2
 - MVS build processor 7.2.2
 - MVS build script 7.2.2.2
- N**
 - naming convention 3.9 6.3.1
 - nonfile parts 6.7
 - notification list 2.1
 - notification server 5.8
 - NULL builder 7.8
 - NVBridge utilities 8.3.4
 - NVBridge/2 2.5.1 2.5.2 8.0
 - NVDM/2 2.5.1
- O**
 - ObjectStore database 4.4
 - ObjectStore server 4.4
 - ObjectStore server parameters 4.4
 - ObjectStore utilities
 - osbackup 11.3 11.3.1
 - oscp 11.3 11.3.2
 - osrestor 11.3 11.3.3
 - ossvrpin 11.3 11.3.4
 - ossvrsta 11.3 11.3.5

- osverify 11.3 11.3.6
- opening a defect 9.3.3.1
- opening a feature 9.2.2.1
- organizing build agents 7.4
- organizing build environments 7.3
- organizing build pools 7.3
- organizing build processors 7.4
- organizing components 2.1
- out-of-date objects 7.9
- output file 7.9
- output managed objects 2.4.1
- P**
- package file 8.1.1 8.1.2 8.2 8.3.1
- packaging 8.0
- packaging and distribution support 2.5
- packaging utilities 2.5.1
- parallel builds 7.4
- parser 2.4.1 7.0 7.6
- parser build script 7.2.3
- part
 - check in 6.7 6.7.3
 - check out 6.7 6.7.1
 - edit 6.7 6.7.2
 - extract 6.7 6.7.6
 - lock 6.7 6.7.4
 - unlock 6.7 6.7.5
- passing parameters 7.2.2.2
- path name 3.9
- path name
- placeholder part 6.6.3
- planning the build environment 3.8
- Pool page 6.1.2.5
- preparing to install 4.1
- problem tracking 2.6 9.0
- process of building 2.4
- processes
- product identification 14.2 14.2.2.2
- project administrator 7.1
- project status 2.7
- pseudotarget 7.8
- Q**
- quality analysis 2.7
- R**
- reconcile 10.1 10.1.2 10.1.3
- reconciling collision records 10.1 10.1.3
- reinstalling ObjectStore 4.4
- reject 9.2.1 9.2.2.7 9.3.1 10.1.3
- relationships between parts 6.6.2
- release
 - concurrent development 3.6.1
 - logical organization 2.2
 - naming convention 3.6.2
 - objects 3.6
 - separate database 3.6.3
 - serial development 3.6.1
- release management 2.2
- release process
 - approval subprocess 6.5.2
 - driver subprocess 6.5.2
 - fix subprocess 6.5.2
 - test subprocess 6.5.2
 - track process 9.0
 - track subprocess 6.5.2
- release processes 5.6
- release subprocess
- reloading the authority table 12.1.2
- reloading the config table 12.4.1.1
- reloading the configurable process tables 12.3.2
- reloading the interest table 12.2.2
- report facility 2.7
- repository and model support 2.9
- repository architecture 2.13
- repository services 2.13
- resolving TeamConnection errors 11.2
- retain lock 6.7.2 6.7.3
- S**
- sample build scripts
 - fhbcob2 7.2.1
 - fhbcob2l 7.2.1
 - FHBCOBM 7.2.2.3
 - FHBMA SM 7.2.2.3
 - FHBMC 7.2.2.3
 - FHBMC370 7.2.2.3
 - FHBMLINK 7.2.2.3

- FHBMPLI 7.2.2.3
- fhbocomp 7.2.1
- fhbolib 7.2.1
- fhbolin2 7.2.1
- fhbolink 7.2.1
- fhborc 7.2.1
- fhbplbld 7.2.1
- FHBPLKED 7.2.2.3
- fhbpllnk 7.2.1
- for C++ compile on OS/2 7.2.1
- Gather/2 8.0
- NVBridge/2 8.0
- nvbridge.cmd 8.3
- schema evolution 2.12
- semantic classes 2.10
- serial development 6.7.1
- server machine 2.4.2
- services file 4.1
- Settings notebook 6.1.2
- Setup page 6.1.2.2
- sizing record 9.4
- sizing records 9.2.1.1 9.2.2.3
- software requirements 3.1.2
- source root directory 8.3.1
- source work area 10.1.3
- specifying a file mode 6.6.3
- stanza view 12.5.1
- start family server 5.8
- start the family server 4.5.2
- start the NVBridge tool 8.3.1
- start the ObjectStore server 4.4
- start the TeamConnection client 4.6
- starting a build agent 7.4.1.3
- starting a build processor on MVS 7.4.1.2
- starting a build processor on OS/2 7.4.1.1
- starting the TeamConnection client 6.1
- starting your build server 7.4.1
- states
 - defects 9.2.1
 - driver 9.3.2
 - features 9.2.1
 - work area 9.3.1
- stop family server 5.8
- storage view 2.9
- superuser 2.1
- T**
- table view 12.5.1
- target root directory 8.2
- target tree 8.2.1.1
- TCMERGE utility 10.1.3
- TCP/IP administrator 4.1
- teamagnt 7.4.1.3
- TEAMC10.INI 6.1.1
- teamcd 7.4.1.3
- TeamConnection
 - access 3.3
 - access authority 3.5.1
 - access list 2.1 6.2.1
 - add a host name 6.2.1
 - add driver members 9.3.3.10
 - adding a driver member 9.3.3.10
 - aging utilities 11.1
 - application development 1.2
 - application development challenges 1.1
 - approval record 9.3.1 9.3.3.7
 - approval subprocess 9.3.3.7
 - approver lists 9.3.1
 - authority groups 5.2 9.1 9.4 12.1.1
 - authority levels
 - explicit authority 3.5.1
 - implicit authority 3.5.1
 - superuser authority 3.5.1
 - backup and recovery 2.8
 - base name 3.9 6.6.3
 - break common link 6.7.2 6.7.3
 - build 3.8
 - build administrator 7.0
 - build agent 2.4.2 3.1.1 7.3 7.4
 - build environment 3.8 7.3
 - build event 2.4.1 7.3 7.7
 - build pool 7.3
 - build processor 2.4.2 3.1.1 7.4
 - build scope 2.4.2
 - build script 2.4.1 7.0 7.2

- build structure 2.4
- build topology 7.4
- build tree 2.4.1 7.0 7.7
- build tree for packaging 8.1
- build tree for the Gather/2 tool 8.1.1
- build tree for the NVBridge/2 8.1.2
- builder 2.4.1 7.0 7.1 7.5
- builder parameters 7.2.1
- builders 3.8
- change control 2.6 9.0
- change control process 9.0
- change process 2.2
- check in 2.3
- check out 2.3
- client 3.1.1
- collision -reconcile 10.1.3
- collision record 10.1 10.1.1 10.1.2 10.1.3 10.1.4
- command line interface commands 6.2.2
- commit 2.3
- common actions 2.1
- common releases 6.7.2 6.7.3
- component
 - access list 3.5.1 6.3
 - assign a new owner 6.3.2
 - evolution 3.4.4
 - naming convention 3.4.5
 - notification list 6.3
 - ownership 3.5
 - project data 6.3
- component hierarchy 2.1 3.4 3.4.1
 - multiple parents 3.4.3
 - parallel hierarchy 3.4.2
 - product organization 3.4.1
- component process 3.7.1
- component processes 5.5 9.0
- component structure 6.3.1
- concurrent development 6.7.1 10.1
- configurable fields 5.4
- configurable process 2.6
- create components 6.3 6.3.1
- create new user IDs 6.2.2
- create releases 6.4.2
- creating a driver 9.3.3.10
- creating a release 6.4
- creating a user 6.2
- creating the family 5.0
- customer value 1.3
- defect 2.6 9.0
- defect and feature states 9.2.1
- delete release 6.4.2
- delivers 1.2
- dependencies 7.1
- developer 7.1
- distributed build 2.4.2
- distribution 8.0
- driver 9.3.2 9.3.3.10 9.3.3.11
- driver -complete 9.3.3.12
- driver member 9.3.3.8
- driver members 9.3.2 9.3.3.10 9.3.3.11
- driver process 2.6
- driver subprocess 9.3.2
- driver subprocesses 9.3.1
- electronic software distribution 2.5.2
- environment list 9.3.3.12 9.3.3.13
- extended JCL syntax 7.2.2.1
- family 2.1 3.2
- family administrator 12.4
- family planning 3.2
- feature 2.6 9.0
- fix records 9.2.2.3 9.3.1 9.3.3.3 9.3.3.9
- host list 3.3 6.2.1
- host list entries 6.2.1
- host name 6.2.1
- information model
 - conceptual view 2.9
 - data constraints 2.11
 - logical view 2.9
 - schema evolution 2.12
 - semantic classes 2.10
 - storage view 2.9
- initial superuser 5.1
- input managed objects 2.4.1
- installation options 4.2.2
- installation planning 3.1

- installation program 4.2
- integrated build 2.4
- integrated problem tracking 2.6 9.0
- integrates 1.2
- interest groups 5.3 9.1 12.2.1
- job queue 2.4.2
- maintain a component structure 6.3.3
- managed object 2.1
- modeled objects 2.1
- modify releases 6.4.2
- multiple families 3.2
- naming convention 3.9 6.3.1
- notification 3.5.2
- notification list 2.1 3.5.2
- notification server 5.8
- NULL builder 7.8
- opening a defect 9.3.3.1
- organizing components 2.1
- output managed objects 2.4.1
- package file 8.1.1 8.1.2 8.2 8.3.1
- packaging 8.0
- packaging and distribution support 2.5
- packaging utilities 2.5.1
- parser 2.4.1 3.8 7.0 7.1 7.6
- part
 - build 6.6
 - check in 6.6
 - check out 6.6
 - collection of data 6.6
 - common part 6.6.1
 - create 6.6
 - extract 6.6
- path name 3.9 6.6.3
- planning 3.2
- predefined authority groups 9.1
- preparing to install 4.1
- problem tracking 2.6
- processes 3.7
- project administrator 7.1
- provides 1.2 1.3
- re-create releases 6.4.2
- release 2.2 3.6.1
- release process 9.0
- release processes 3.7.2 5.6 6.4.2
- release-level versioning 2.3
- report facility 2.7
- reports 10.3
- repository and model support 2.9
- repository architecture 2.13
- repository services 2.13
- retain lock 6.7.2 6.7.3
- sample build scripts 7.2.1 7.2.2.3
- search for components 6.3.1
- serial development 6.7.1
- server 3.1.1
- server machine 2.4.2
- sizing records 9.2.2.3 9.2.2.4 9.3.3.3
- start family server 5.8
- stop family server 5.8
- superuser 2.1 3.3
- supports 1.2
- terminology 2.0
- test records 9.3.1 9.3.3.13
- test subprocess 9.3.1 9.3.2 9.3.3.12 9.3.3.13
- text objects 2.1
- tools integration 2.13
- topology 3.8 7.1
- trace environment variables 11.2.3
- track process 2.2 2.6
- track subprocess 9.3.2
- user exits 5.7
- user ID 3.3 6.2.1
- user list 3.3
- verification records 9.2.2.7 9.3.3.14
- verify subprocess 9.3.2 9.3.3.14
- versioning 2.3 6.8
- versioning model 2.3
- work area 2.3 6.5 6.5.1 6.5.2
 - See also work area
- workarea -refresh 10.1.3
- working with defects and features 9.2
- working with parts 6.7
- writing a package file 8.2.2.1 8.3.2 8.3.2.2

TeamConnection - Tasks window 6.1.1

TeamConnection commands
 approval -accept 9.3.3.7
 builder 7.5 7.8
 chfield 12.4.2 12.4.2.1
 defect 9.3.3.1 9.3.3.2 9.3.3.3
 defect -accept 9.3.3.5
 defect -review 9.3.3.4
 driver -commit 9.3.3.11
 driver -create 9.3.3.10
 drivermember -create 9.3.3.10
 feature 9.2.2.1 9.2.2.2 9.2.2.3 9.2.2.4 9.2.2.5 9.2.2.7
 fhclproc 12.3.1 12.3.2
 fix -complete 9.3.3.9
 host 6.2.2
 parser 7.6
 part 6.6.3 6.7.3 6.7.4 6.7.5 6.7.6
 parts 6.7.1
 release 6.4.2
 report 6.7.1
 size -accept 9.3.3.3
 size -create 9.3.3.3
 test -accept 9.3.3.13
 user 6.2.2
 verify 9.2.2.7
 verify -accept 9.3.3.14
 workarea 6.5.2 6.8
 workarea -integrate 9.3.3.9
TeamConnection merge 10.1.4
TeamConnection subprocesses 9.4
TeamConnection variables
 &TCINPUT 7.2.2.2
 &TCOUTPUT 7.2.2.2
 &TCPARM 7.2.2.2
 &TCRELEAS 7.2.2.2
 &TCWORKAREA 7.2.2.2
teamcpak 8.2.1.2
teamcpak nvbridge 8.3.1.2
terminology 2.0
test record 9.4
test records 9.3.1
text files 6.6.3
text objects 2.1
 the defect in the working state 9.3.3.6
 the work area in the approval state 9.3.3.7
 the work area in the fix state 9.3.3.8
tools integration 2.13
topologies 2.4.2
trace environment variables
 BSERV_LOG 11.2.3
 BSERV_TRACEATTEMPTS 11.2.3
 BSERV_TRACEDELAY 11.2.3
 BSERV_TRACEFILE 11.2.3
 BSERV_TRACESIZE 11.2.3
 TC_TMP 11.2.3
 TC_TRACE 11.2.3
 TC_TRACEATTEMPTS 11.2.3
 TC_TRACEDELAY 11.2.3
 TC_TRACEFILE 11.2.3
 TC_TRACESIZE 11.2.3
U
unauthorized transactions 11.2.2
unit builds 7.0
unlock 6.7
updating configurable fields 12.4.2.3
user exit programs 10.4.1 10.4.2
user exits 5.7 10.4 12.6
user list 3.3
user roles
 actions they can perform 9.1.2
 builder 9.1.2
 commands they can use
 approval 9.1.1
 approver 9.1.1
 coreq 9.1.1
 defect 9.1.1
 driver 9.1.1
 drivermember 9.1.1
 feature 9.1.1
 fix 9.1.1
 size 9.1.1
 test 9.1.1
 verify 9.1.1
 workarea 9.1.1
 componentlead 9.1.2

- developer 9.1.2
- developer+ 9.1.2
- projectlead 9.1.2 9.3.3.10
- releaselead 9.1.2
- writer 9.1.2
- writer+ 9.1.2
- using a NULL builder 7.8
- using TeamConnection to package products 8.0
- using the audit log 11.2.2
- using the error log 11.2.1
- using the Gather/2 tool 8.2
- using the merge tool 10.1.4
- using the NVBridge/2 Tool 8.3
- using the preship process 9.3.3
- using the teamcpak command 8.2.1 8.3.1
- using the trace facility 11.2.3

V

- verification record 9.4
- verify TeamConnection installation 4.5.3
- version control 14.2.2
- versioning 2.3 6.8
- versioning model 2.3
- view sizing information 9.3.3.3

W

- work area
 - change the state 6.5.2
 - committing 6.5
 - commonality 6.7.3
 - create 6.5 6.5.2
 - definition 6.5
 - delete 6.5.2
 - freeze 6.5 6.5.2 6.8
 - freezing 6.5
 - integrate 6.8 10.1.3
 - linking 6.5
 - modify 6.5.2
 - reassign 6.5.2
 - refresh 6.5.2 10.1.3
 - refreshing 6.5
 - specified name 6.5.2
 - view information 6.5.2
- work area states
 - approval 9.3.3.7
 - approve 9.3.1
 - commit 9.3.1 9.3.3.11
 - complete 9.3.1 9.3.3.12 9.3.3.13 9.3.3.14
 - fix 9.3.1 9.3.3.8
 - integrate 9.3.1 9.3.3.9 9.3.3.11
 - test 9.3.1 9.3.3.12 9.3.3.13
- working with defects and features 9.2
- working with parts 6.7
- workload balancing 2.7
- writing a package file
 - keywords 8.2.2.1 8.3.2.2
 - syntax rules 8.2.2.1 8.3.2.1
- writing user exit 10.4.1